INPUT

INPUT
INPUT
INPUT
INPUT
INPUT
INPUT
INPUT
INPUT
INPUT
INPUT
INPUT
INPUT
INPUT
INPUT
INPUT
INPUT
INPUT
INPUT

1

SUMÁRIO

APLICAÇÕES

CÓDIGO DE MÁQUINA

PERIFÉRICOS

PROGRAMAÇÃO BASIC

PROGRAMAÇÃO DE JOGOS

PROGRAMAÇÃO EM CÓDIGO DE MÁQUINA

- O QUE É CÓDIGO DE MÁQUINA?
- VANTAGENS EM RELAÇÃO AO BASIC.
- COMO COMPREENDER CÓDIGO DE OPERAÇÃO E LINGUAGEM ASSEMBLER.

Utilizado na programação de jogos, o código de máquina proporciona uma ação rápida e contínua. Antes de empregá-lo, porém, você deve saber como ele afeta o desempenho do seu computador.

O BASIC constitui sem dúvida a mais difundida e popular linguagem de programação. Universalmente conhecido, ele é fácil de aprender e pode ser adaptado a diferentes máquinas. Seus programas, contudo, ocupam grandes espaços de memória e permitem apenas um movimento de cada vez. Assim, se um canhão atira, num jogo de guerra, o resto da ação tem que esperar, mesmo que seja por uma fração de segundo.

O BASIC usa palavras da linguagem humana, tiradas do inglês, e operações semelhantes às da aritmética, fáceis de compreender. Mas seu computador não pensa em inglês nem entende os símbolos aritméticos. Ele opera baseado em impulsos elétricos que representam números. E essa conversão é a causa de sua lentidão ao trabalhar com o BASIC.

Código de máquina é uma linguagem computacional composta apenas de números equivalentes àqueles que o computador utiliza. Assim, quando você emprega esse código não deve esperar que o computador responda em linguagem humana.

Consideremos um exemplo extraído dos computadores compatíveis com o Sinclair Spectrum; o código de máquina se parecerá com isto:

```
B9 28 08
```

Em BASIC o equivalente é:

```
100 IF A=C THEN GOTO 190
```

O código de máquina consiste, desse modo, numa série de números de dois dígitos. A letra B da linha acima é, na realidade, um símbolo. Ela representa o número 11, em notação hexadecimal.

Esses números hexadecimais são introduzidos na memória do computador. Instruções de operação, dados, números, letras, palavras e endereços de memória são representados por sinais de dois dígitos. E o computador identifica a diferença entre essas informações pela ordem em que eles ocorrem no programa. Por exemplo, o primeiro número em qualquer programa precisa representar uma instrução. Se, por engano, você digitar um número qualquer nesse ponto, representando um dado ou um endereço de algum dado, o computador tentará interpretá-lo como uma instrução válida. A digitação desses números exige, portanto, uma precisão absoluta; caso contrário, o programa não funcionará.

VANTAGENS DO CÓDIGO DE MÁQUINA

Quando você digita uma linha em BASIC, o computador tem que transpô-la para sua própria linguagem, antes de executá-la. Este é um processo trabalhoso que toma muito tempo, pois uma instrução em BASIC raramente é traduzida para um só comando ou declaração em código de máquina. Ela resulta, com freqüência, em vários códigos de operação.

Quando se executa um programa completo escrito em BASIC, cada linha tem que ser interpretada seqüencialmente. O computador não armazena os resultados dessa tradução; se a mesma linha for executada novamente, o computador terá que interpretá-la outra vez.

Sempre que o computador encontra uma instrução em BASIC, executa as seguintes tarefas:

- reconhecer a instrução em BASIC;
- traduzir essa instrução para uma série de outras em código de máquina;
- executar as instruções, uma a uma;
- passar para a próxima linha do programa em BASIC.

Repetido muitas vezes, esse processo torna-se lento e moroso. É possível, porém, compilar um programa em BASIC, isto é, traduzi-lo integralmente para código de máquina, de uma só vez, antes de executá-lo, armazenando a tradução. A compilação é mais eficiente

que a interpretação, mas os programas resultantes são ainda bastante lentos.

Em contrapartida, a tradução do programa em código de máquina diretamente para a operação interna do computador quase não consome tempo. Nesse caso, há uma simples conversão de um número em outro, e não a tradução de um comando de linguagem em vários outros de código de máquina. Os programas tornam-se, assim, mais curtos e eficientes.

Os programas abaixo fazem a mesma coisa, usando BASIC e código de máquina. Compare suas velocidades relativas.

```
10 CLEAR 200,31000
20 DEFUSR0=31000
30 FOR N=31000 TO 31015
40 READ A
50 POKE N,A
60 NEXT
70 CLS0
80 PRINT @0,"ISTO ESTA EM BASIC
 "
90 FOR N=1 TO 500:NEXT
100 FOR N=1056 TO 1535
110 POKE N,PEEK(N+34000)
120 NEXT
130 FOR N=1 TO 1000:NEXT
140 CLS0
150 PRINT @0,"ISTO ESTA EM CODI
GO DE MAQUINA."
160 FOR N=1 TO 1000:NEXT
170 N=USR0(0)
180 FOR N=1 TO 2000:NEXT
190 DATA 206,136,240,142,4,32,1
66,192,167,128,140,6,0,38,247,5
7
```

Atenção: o programa abaixo está escrito em BASIC para computadores com o sistema DOS (disquete) e não será aceito pelo BASIC nível II normal (cassete).

Ao ligar a máquina, responda com o número 60000 à pergunta "Mem.usada?", ou ainda "Memory size?" (para computadores com 48 Kbytes de memória apenas).

```
10 CLS
20 DEFUSR0=-4536
30 FOR N=-4356 TO -4523
40 READ A
50 POKE N,A
60 NEXT N
70 CLS
80 PRINT "ISTO ESTA  EM BASIC"
90 FOR I=1 TO 500:NEXT
100 FOR N=15360 TO 16383
110 POKE N,65
120 NEXT N
130 FOR N=1 TO 1000:NEXT
140 CLS
150 PRINT  "ISTO ESTA' EM  LIN-
    GUAGEM DE MAQUINA"
160 FOR N=1 TO 1000:NEXT
170 N=USR0(0)
180 FOR I=1 TO 2000:NEXT
190 DATA 33,0,60,17,1,60,1,255,
    3,54,65,237,176,201
```

```
10 CLEAR 29999
20 FOR n=30000 TO 30011
30 READ a
40 POKE n,a
50 NEXT n
60 PRINT "Isto esta em BASIC"
70 FOR n=16384 TO 22527
80 POKE n,PEEK (n-16384)
90 NEXT n
100 CLS
110 PRINT "Isto esta em Codigo
 de Maquina"
120 PAUSE 100
130 RAND USR 30000
140 STOP
150 DATA 33,0,0,17,0,64,1,0,24
,237,176,201
```

```
10 CLEAR200,&HDFFF
20 DEFINTA-Z
30 AD=&HE000:DEFUSR=AD
40 FOR I=0TO10
50 READA$
60 POKE AD+I,VAL("&H"+A$)
70 NEXT
80 DATA0E,98,ED,6B,0,1,06,FF,ED
,B3,C9
90 CLS
100 PRINT "Isto esta em BASIC"
110 FOR I=1 TO 1000:NEXT
120 SCREEN 2
130 PSET (0,0)
140 FOR I=1 TO 6144
150 OUT 152,RND(1)*255
160 NEXT I
170 CLS
180 SCREEN 0
190 PRINT"Isto esta em código d
e máquina"
200 FOR I=0 TO 1000:NEXT
210 SCREEN 2
220 PSET(0,0)
230 FOR I=1 TO 25
240 A=USR(B)
250 NEXT
260 GOTO260
```

```
5   HGR : HOME : VTAB 24
10   FOR I = 800 TO 836
15   READ N
20   POKE I,N
25   NEXT
30   PRINT "Isto esta em BASIC"
35   FOR I = 1 TO 300: NEXT
40   FOR I = 8100 TO 14000
45   POKE I, RND (1) * 256
50   NEXT
55   TEXT : HGR : HOME : VTAB 24
: PRINT "Isto esta em linguagem
 de maquina"
60   FOR I = 1 TO 3000: NEXT
65   CALL 800
70   DATA  169,0,133,20,133,22,1
69,32
80   DATA  133,21,169,193,133,23
,160,255
90   DATA  177,22,145,20,136,208
,249,165
100   DATA  21,201,63,208,1,96,2
30,21,230,23,76,46,3
```

Como você vai ver, tudo que o programa faz é encher a tela de caracteres ao acaso ("lixo", na gíria dos programadores). Entretanto, a velocidade com que a versão em linguagem de máquina faz isso é incomparavelmente maior do que a do programa em BASIC.

LINGUAGEM ASSEMBLER

A grande dificuldade do código de máquina surge quando se quer escrevê-lo ou depurá-lo de erros. Poucas pessoas conseguem lembrar-se de todos os códigos numéricos e instruções. Para complicar ainda mais, os códigos de operação (opcodes) não são distinguíveis dos outros números que alimentam o computador. Assim, você não consegue entender um trecho de programa, a não ser que o acompanhe desde o começo — o que não ajuda quando se está procurando erros no programa.

Os códigos numéricos de operação, além disso, diferem consideravelmente entre si, conforme o microprocessador que é usado no computador, de modo que traduzir programas em código de máquina de um tipo de computador para outro pode ser bastante difícil.

Uma forma de contornar esses obstáculos é subir um pouco mais de nível, e escrever o programa em uma linguagem mais fácil de se utilizar do que o có-

digo de máquina. Essa linguagem é conhecida como Assembly, ou linguagem Assembler (que vem do inglês, montagem).

Em Assembler, os códigos operacionais são representados por abreviações mnemônicas (isto é, fáceis de lembrar). Por exemplo, a operação de carregar (load, em inglês) uma memória com um número, tem a abreviatura **LD.** Uma instrução de desvio (jump) pode ser chamada de **J**, **JP**, ou **JMP**, conforme a sintaxe do Assembler em uso. Com algum treino, é possível ler programas em Assembly tão facilmente quanto em BASIC, embora entendê-los seja algo mais complicado.

A desvantagem da linguagem Assembler é que o computador não pode utilizá-la diretamente, como no caso do código de máquina. Antes disso, o programa precisa ser montado por um Assembler, ou tradutor, que é um outro programa escrito em linguagem de máquina, ou mesmo em BASIC.

Mas o processo de montagem é bem mais simples, quando comparado com a interpretação de linhas em BASIC. A linguagem Assembler é equivalente ao código de máquina, ou seja, cada instrução em Assembly corresponde a uma instrução em código de máquina. Assim, a tradução do programa se processa palavra por palavra, número por número, etc. diretamente para código de máquina.

O Assembler também traduz o programa como um todo, antes que o computador o execute, ao invés de interpretá-lo linha por linha, enquanto o programa está rodando. Isto dá vantagens adicionais de velocidade de execução.

LDA #20
STA X+
ANDB #127
LSLB

Alguns computadores, como o Apple II, já vêm com um programa embutido, chamado monitor, que permite a entrada de códigos de máquina e dados em hexadecimal. Computadores com programas Assembler embutidos em sua ROM, entretanto, já são mais raros. No Brasil, o TK-2000, da Microdigital, é um exemplo. Para os outros computadores (linha TRS-80, TRS-Color, MSX, Sinclair Spectrum e ZX-81), existem programas Assembler disponíveis comercialmente, em fita ou disquete. Você poderá utilizar também, as versões de Assembler para cada máquina, que serão fornecidas mais adiante.

Mas, se você não tem acesso a um programa Assembler, pode fazer uma montagem manual. Mesmo para os programas mais simples, é mais fácil escrevê-los em Assembly, e depois traduzi-los manualmente para código de máquina em hexadecimal, usando as tabelas fornecidas com os manuais de programação Assembler.

Tudo isto pode parecer muito aborrecido; mas, antes de decidir que é isso mesmo, experimente esses programas em código de máquina. Eles são introduzidos via declarações **DATA** em programas BASIC. Os dados (**DATA**) estão listados no fim de cada programa.

```
10 CLEAR 200,31000
20 DEFUSR0=31000
30 FOR N=31000 TO 31020
40 READ A
50 POKE N,A
60 NEXT
70 N=USR0(0)
80 POKE 32767,RND(256)-1
90 FOR N=1 TO 100:NEXT
100 GOTO 70
110 DATA 142,4,0,206,136,184,16
6,192,184,127,255,138,129,167,1
28,140,6,0,38,242,57
```

As restrições na configuração da máquina para rodar o programa abaixo são as mesmas do primeiro programa.

```
20 DEFUSR0=-4536
30 FOR N=-4536 TO -4525
40 READ A
50 POKE N,A
60 NEXT N
70 N=USR(0)
80 POKE -4534,RND(20)
90 GOTO 70
100 DATA 33,0,0,17,0,60,1,0,4,
    237,176,201
```

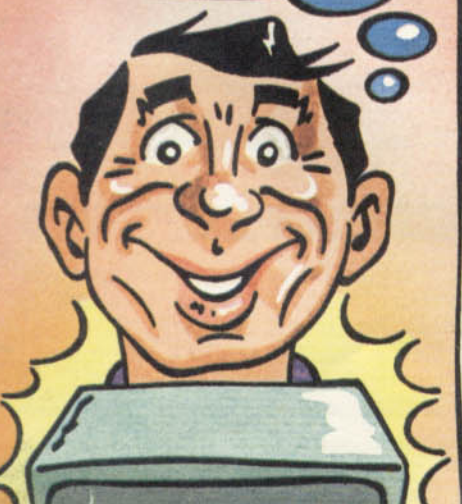
```
10 CLEAR 29999
20 FOR n=30000 TO 30020
30 READ a
40 POKE n,a
50 NEXT n
60 RAND USR 30000
70 GOTO 60
80 DATA 17,0,88,46,0,237,95,
71,58,140,92,128,230,63,103,1
,0,3,237,176,201
```

```
10 CLEAR200,&HDFFF
20 DEFINTA-Z
30 AD=&HE000:DEFUSR=AD
40 FOR I=0TO10
50 READA$
60 POKE AD+I,VAL("&H"+A$)
70 NEXT
80 DATA0E,98,ED,6B,0,1,06,FF,ED
,B3,C9
90 FOR I=0 TO 1000:NEXT
100 SCREEN 3
110 PSET(0,0)
120 A=USR(B)
130 GOTO 120
200 FOR I=0 TO 1000:NEXT
210 SCREEN 3
220 PSET(0,0)
240 A=USR(B)
260 GOTO 240
```

Um aviso para quem for utilizar o programa acima: o programa em linguagem de máquina altera alguns endereços importantes da memória de trabalho do micro. Por isso, é impossível listá-lo depois de rodar uma vez. Assim, você deve digitar o programa e gravá-lo em fita ou disquete, antes de rodá-lo.

Para rodar no TK-2000, é necessário fazer as seguintes modificações: todos os valores 17, que aparecem dentro das declarações DATA, devem ser mudados para 20, e todos os valores 18, para 21. Deste modo ele não se autodestrói.

```
10  FOR I = 800 TO 840
20  READ N
30  POKE I,N
40  NEXT
50  GR
60  CALL 800
70  DATA  169,0,133,17,169,193,
133,18,160
80  DATA  40,162,40,177,17,41,1
5,32,100
90  DATA  248,138,32,0,248,136,
208,242,202
100  DATA  208,239,169,254,197,
18,208,1,96,230,18,76,40,3
```

Agora, apenas para comparar, tente escrever um programa similar em BASIC. Acreditamos que você entenderá nossas razões. E mais tarde, veremos o que acontece aqui.

ANIMAÇÃO E SINAIS GRÁFICOS

Se você quer dar mais vida aos seus jogos, comece a trabalhar com os caracteres gráficos mais simples, que já vêm programados na memória do seu computador.

Programar jogos pode se revelar uma atividade fascinante. Mas não é certamente das mais fáceis. Por isso, você deve começar com coisas simples e ir avançando pouco a pouco. Esse procedimento lhe ensinará a pensar de maneira lógica, melhorando suas qualidades de programador. É o que vamos aprender nesta série de lições, até chegar ao desenvolvimento de jogos profissionais e complexos.

A primeira coisa que você precisa aprender para programar jogos, afora a linguagem BASIC, é a técnica de animação, pois a maioria dos jogos mais interessantes é do tipo "videogame", isto é, tem algum tipo de ação.

Para criar a ilusão de movimento, o programador usa praticamente as mesmas técnicas empregadas na elaboração de desenhos animados. Ele cria duas ou mais imagens e as alterna rapidamente — cerca de 24 vezes por segundo.

Existe, porém, uma diferença importante. Na animação de um desenho, o projetor é responsável pela eliminação das imagens desnecessárias. O mesmo não acontece com a animação por computador: neste caso, qualquer segmento de um desenho que você "projete" permanecerá na tela, a não ser que você imprima alguma coisa sobre ele, pois o computador não pode colocar, simultaneamente, duas imagens na mesma posição.

Por exemplo, se a linha 10 diz ao computador para colocar um *A* em um determinado lugar, qualquer linha impressa posteriormente — digamos um *B* —, no mesmo local, apagará o *A*.

Mas, e se não houver nada que você queira imprimir no lugar do caractere indesejado? Então você deve se lembrar de incluir no seu programa uma linha que coloque um espaço em branco na posição de interesse. Caso isso não seja feito, seu vídeo logo estará repleto de incômodos pedaços de braços, pernas e corpos!

Há grandes diferenças entre os computadores na maneira de se obter os caracteres (sinais) gráficos na tela. Os caracteres gráficos padronizados variam muito em tipo e disposição, assim como a maneira como são impressos na tela, e como são movimentados.

Uma animação muito simples, que ilustra os princípios da produção de efeitos de movimentação na tela do micro, é a do pequeno "inseto rastejante" (veja abaixo). Nas páginas seguintes mostraremos como conseguir esse efeito para diversos tipos de computadores.

```
 )))      )))      )))      )))
 OOO <    OOO <    OOO <    OOO <
 )))      )))      )))      )))
```

T

Para criar a figura do inseto na tela, programe o TRS-Color ou compatível (por exemplo, o Prológica CP-400), como segue abaixo. A declaração **PRINT** @ (pronuncia-se "print arroba" ou "print em") é a maneira que o BASIC do Color tem para exibir um ou vários caracteres em um ponto específico da tela. O número que se segue após o sinal @ é a posição na tela, que começa de 0, no canto superior direito, e aumenta de um em um da esquerda para direita e de cima para baixo. Como a tela do Color tem apenas 32 colunas, a primeira linha tem posições @ indo de 0 a 31. A posição 32 já se situa na primeira coluna da segunda linha e assim por diante.

```
5 CLS
10 PRINT @238,"OOO"
20 PRINT @206,")))"
30 PRINT @241,"<"
40 PRINT @270,")))"
```

Este programa é uma forma bastante elaborada de criar uma imagem muito simples, mas ele ilustra alguns pontos interessantes:

Em primeiro lugar, ele lhe dá uma idéia a respeito das posições relativas na tela. O meio do inseto está, aproximadamente, no centro do vídeo, na localização 239. Como você vê, estamos usando caracteres "normais" (isto é, disponíveis no próprio teclado padrão, como a letra O, o sinal de parênteses, etc.) para compor a figura do "bicho".

Além disso, ele mostra o que acontece quando fazemos com que o computador imprima mais que um caractere em uma única posição de tela — ele simplesmente vai em frente e coloca os próximos caracteres nas posições subseqüentes (de numeração maior). Isto explica por que as "antenas" da linha 30 estão em 241. As posições 238-240 já estão ocupadas pelo corpo do inseto.

Existe uma maneira melhor de desenhar o inseto, que é condensar o programa acima em uma única linha. E, se você ainda não descobriu, o TRS-Color tem uma abreviação para **PRINT**: o caractere "?". Quando o programa é listado, a máquina mostrará **PRINT** ao invés de "?".

O programa simplificado fica assim:

```
20 PRINT @238,"OOO<":PRINT @206
,")))":PRINT @270,")))"
```

Se você acrescentar mais duas linhas, obterá alguma animação:

```
 )))      )))      )))      )))
 OOO <    OOO <    OOO <    OOO <
 )))      )))      )))      )))
20 PRINT @238,"OOO<":PRINT @206
,"(((":PRINT @270,"((("
30 GOTO 10
```

A execução deste programa produz uma imagem tremida e pouco nítida que não se parece nem um pouco com animação. A razão disso é que as imagens estão sendo trocadas rápido demais. Se você inserir um laço **FOR...NEXT** no programa, ele fará com que o computador pare e conte, deixando a animação mais clara. Qualquer número pode ser usado com os laços **FOR...NEXT** (linhas 15 e 25) e não apenas 15. Outros números resultarão em uma espera maior ou menor.

Experimente adicionar estas linhas:

```
15 FOR L=1 TO 15
17 NEXT L
25 FOR L=1 TO 15
27 NEXT L
```

Agora você tem um inseto que esperneia incessantemente, um tanto sem razão; apesar disso, a tela nos mostra uma animação convincente.

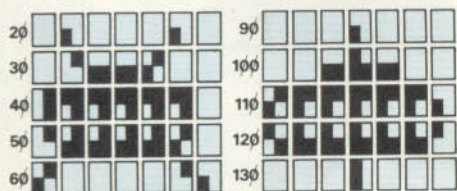
GRÁFICOS DE BAIXA RESOLUÇÃO

Você pode usar os caracteres gráficos de baixa resolução do seu computador para criar figuras animadas mais interessantes. O Manual do Usuário mostra quais são esses caracteres. Cada um deles tem um código (um valor de 128

- MOVIMENTE AS FIGURAS NO VÍDEO COMO NUM DESENHO ANIMADO
- APRENDA A USAR CARACTERES GRÁFICOS

a 143 para os micros da linha TRS-Color). Para mostrá-los no vídeo você usa a função **CHR$** seguida do código do caractere entre parênteses.

Por exemplo, para imprimir o caractere representado pelo código 138 no meio da tela, digite:

```
10 PRINT@ 239, CHR$(138)
```

Como construir o seu satélite.

O programa seguinte utiliza caracteres gráficos de baixa resolução para animar um satélite rotatório (a fig. 1 mostra os dois desenhos que são alternados no programa).

```
10 CLEAR 500:CLS
20 PRINT @174,CHR$(143)+CHR$(14
1):PRINT@178,CHR$(143)
30 PRINT @206,CHR$(140)+CHR$(13
2)+CHR$(141)
40 PRINT @236,CHR$(137)+CHR$(12
9)+CHR$(129)+CHR$(129)+CHR$(129
)+CHR$(129)+CHR$(141)
50 PRINT @268,CHR$(135)+CHR$(13
2)+CHR$(132)+CHR$(132)+CHR$(132
)+CHR$(133)+CHR$(135)
60 PRINT @300,CHR$(143):PRINT@3
03,CHR$(133):PRINT @305,CHR$(14
3)+CHR$(143)
70 FOR X=1 TO 20
80 NEXT X
90 PRINT @173,CHR$(141):PRINT@1
77,CHR$(141)
100 PRINT @205,CHR$(139)+CHR$(1
40)+CHR$(140)+CHR$(137)
110 PRINT @236,CHR$(138)+CHR$(1
30)+CHR$(130)+CHR$(130)+CHR$(13
0)+CHR$(130)+CHR$(143)
120 PRINT @268,CHR$(139)+CHR$(1
36)+CHR$(136)+CHR$(136)+CHR$(13
6)+CHR$(137)+CHR$(143)
130 PRINT @300,CHR$(137):PRINT@
303,CHR$(143):PRINT @305,CHR$(1
39)+CHR$(141)
140 FOR X=1 TO 20
150 NEXT X
160 GOTO 10
```

Não se preocupe com a linha 10 do programa. **CLEAR 500** reserva espaço na memória para as cadeias de caracteres dos códigos **CHR$** e **CLS**. Examine os sinais gráficos representados pelos códigos após **CHR$** no programa e tente entender como a espaçonave foi construída. A instrução '+ **CHR$** (   )' significa **'PRINT CHR$** (   ) na próxima posição de tela'.

MOVIMENTO

Eis aqui o programa que anima o inseto, movendo-o pela tela:

```
10 CLS
20 FOR N=0 TO 28
30 PRINT @192+N,")))":PRINT @22
4+N,"OOO<":PRINT @256+N,")))"
40 FOR X=1 TO 10
50 NEXT X
60 PRINT @192+N,"    ":PRINT @22
4+N,"    ":PRINT @256+N,"    "
70 PRINT @192+N,"(((":PRINT @22
4+N,"OOO<":PRINT @256+N,"((("
80 FOR X=1 TO 10
90 NEXT X
100 PRINT @192+N,"    ":PRINT @2
24+N,"    ":PRINT @256+N,"    "
110 NEXT N
120 GOTO 20
```

Existem três laços **FOR...NEXT** no programa. Os dois que utilizam **X** diminuem a rapidez de impressão fazendo o computador contar até dez; o que usa **N** faz com que o inseto se mova na tela.

Você pode não ter entendido por que a linha 20 mostra **FOR N = 0 TO 28** se existem 32 posições disponíveis em cada linha da tela. A razão para isso é que o inseto tem quatro espaços de comprimento, e, se houvesse mais que 28 na linha 20, a antena apareceria no lado oposto do vídeo, uma linha abaixo. Isto acontece devido ao modo com que as posições de tela são numeradas. A posição 32, por exemplo, é a primeira da segunda linha, a partir do topo da tela.

As linhas 60 e 100 parecem não estar imprimindo nada; elas são as linhas "apagadoras" descritas anteriormente.

Para os micros da linha TRS-80 os programas acima podem ser modificados com facilidade, bastando alterar os números após os comandos **PRINT @**, já que nesses micros uma linha tem 64 colunas, ao invés das 32 da linha Color.

O meio da tela fica mais ou menos na posição 450. Subtraindo 64 dessa posição temos 386 para a posição inicial das perninhas de cima, e somando 64, obtemos 514 (para as perninhas de baixo). Finalmente, 453 nos dá a posição das antenas. O programa final fica assim:

```
10 PRINT @450,"OOO<":PRINT@386,
   ")))":PRINT@514,")))"
15 FOR L=1 TO 15
17 NEXT L
20 PRINT @450,"OOO<":PRINT@386,
   "(((":PRINT@514,"((("
25 FOR L=1 TO 15
27 NEXT L
30 GOTO 10
```

TRABALHE COM CARACTERES GRÁFICOS

Estude agora com atenção tudo o que está dito na seção sobre o uso de caracteres gráficos no TRS-Color. Ela se aplica exatamente aos micros da linha TRS-80, que têm apenas caracteres gráficos de baixa resolução.

Os truques para desenhar figuras e movimentá-las são iguais aos ensinados para o TRS-Color, ou seja, basicamente, o comando **PRINT @**, e a função **CHR$** usada em conjunto com os códigos numéricos dos caracteres gráficos.

As duas únicas diferenças são as seguintes:

— O conjunto de caracteres gráficos do TRS-80 é bem maior, pois cobre os códigos de 129 a 191. Cada caractere é a combinação de seis pontos em uma grade de três de altura por dois de largura (o TRS-Color possui caracteres gráficos em uma grade de 2 × 2, ou seja, com combinações de quatro pontos por caractere).

— Como dissemos acima, o número de posições @ na tela também é maior: 1024 (dezesseis linhas de 64 colunas cada).

Os efeitos gráficos de baixa resolução no TRS-80 são potencialmente mais ricos e detalhados do que o Color (com apenas uma desvantagem para o primeiro: não podemos escolher a cor dos caracteres gráficos no TRS-80, que tem o vídeo monocromático).

Experimente fazer um programa igual ao do Color, para traçar o satélite (combinação de vários **CHR$**), e depois

128	128	158	128	128
152	131	190	176	128
128	136	157	172	128
128	160	133	170	144
136	145	128	128	128

128	128	187	128	128
168	172	157	142	128
130	174	145	128	128
176	186	130	148	128
129	128	128	181	128

O corredor em duas posições.

faça algo mais difícil: animar um corredor de maratona. A definição da figura do corredor em caracteres gráficos da linha TRS-80 pode ser vista na fig. 2.

Abaixo vemos como programar o pequeno "monstro rastejante" para qualquer micro da linha Sinclair (por exemplo, o TK-85 ou o TK-90X). Para começar, tente criá-lo numa posição estática:

```
10 PRINT AT 10,15;"OOO"
20 PRINT AT 9,15;")))"
30 PRINT AT 11,15;")))"
40 PRINT AT 10,18;"<"
```

O programa usa a declaração **PRINT AT**, que serve para colocar em algum ponto da tela um caractere ou conjunto de caracteres. Os números que se seguem à declaração **PRINT AT** representam o número da linha onde ficará a figura (que vai de 0 a 21), e o número da coluna (que vai de 0 a 31). Assim, **PRINT AT 10,6; "A"**, por exemplo, pede ao computador para escrever a letra **A** na posição definida pela linha 10 e pela coluna 6 da tela (um quadriculado imaginário, como em um jogo de 'batalha naval').

Ele mostra o efeito de ordenar ao computador que imprima mais que um caractere numa mesma posição da tela: ele simplesmente avança e imprime os caracteres seguintes nas posições vizinhas. Esta é a razão pela qual as "antenas" do inseto na linha 40 estão em 10,18. As localizações 10,15; 10,16 e 10,17 já estão ocupadas pelo corpo do inseto.

Uma maneira mais conveniente de compor o inseto consiste em condensar as instruções acima em uma única linha; desta forma (só vale para o TK-90X):

```
10 PRINT AT 10,15;"OOO<";AT 9
,15;")))";AT 11,15;")))"
```

Adicione agora mais duas linhas e você terá animação:

```
20 PRINT AT 10,15;"OOO<";AT 9
,15;"(((";AT 11,15;"((("
30 GOTO 10
```

Quando executar esse programa, você verá que ele produz uma imagem pouco nítida. Isso acontece porque as imagens são trocadas muito rapidamente.

A melhor maneira de desacelerar o processo é usar um laço **FOR...NEXT** que faz com que o computador conte até dez (ou qualquer outro número que você queira). Antes de imprimir a imagem seguinte. Experimente, então, adicionar estas linhas ao seu programa (no TK-85 use 2 ao invés de 10):

```
15 FOR L=1 TO 10
17 NEXT L
25 FOR M=1 TO 10
27 NEXT M
```

Você pode variar a duração da pausa simplesmente mudando **'1 TO 10'** para **'1 TO 5'** ou **'1 TO 20'**, por exemplo.

O inseto criado até agora pode parecer inútil e estúpido, pois esperneia como louco mas não se move. Mais adiante veremos como essa situação se modifica (seção MOVIMENTO).

COMO ANIMAR UMA FIGURA

Uma animação mais interessante pode ser produzida utilizando os caracteres gráficos padrão da linha Sinclair. Temos um exemplo na fig. 2.

O programa completo é dado mais abaixo; no entanto, se você não está habituado aos símbolos gráficos, será interessante criar, inicialmente, uma figura estática, compondo uma linha de cada vez, deste modo:

```
1 PRINT AT 5,15;"O"
2 PRINT AT 6,14;"
```

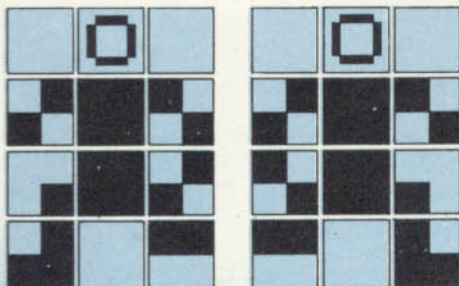
... e assim por diante.

Encontrar esses caracteres gráficos é relativamente fácil. Para obter aqueles da linha 2 acima, por exemplo, tecle **CAPS SHIFT (SHIFT** no TK-85) e **9**, simultaneamente. Isso o coloca no modo gráfico indicado pelo G piscando na tela. Então, no TK-90X, pressione **CAPS SHIFT** e **6** juntos para obter uma versão invertida — preto no lugar do branco — do símbolo da tecla 6; depois **CAPS SHIFT** e **8** e **6** sozinhos. No

TK-85 pressione **SHIFT** e **T**; **SHIFT** e **SPACE**; **SHIFT** e **Y**. Finalmente, tecle **9** para deixar o modo gráfico antes de inserir as aspas no fim da linha.

Aqui temos o programa completo para formar a figura:

```
10 PRINT AT 5,14;"□o□";AT 6,
14;"▞█▚";AT 7.14;"▗█▞";
AT 8,14;"▟□▀"
20 PRINT AT 5,14;"□o□";AT 6,
14;"▞█▚";AT 7,14;"▚█▖";
AT 8,14;"▀□▙"
30 GOTO 10
```

É possível que você sinta necessidade de inserir novamente um laço **FOR...NEXT** depois da linha 10 e outro após a linha 20 para desacelerar a ação.

Um dançarino feito de doze quadrinhos.

MOVIMENTO

Agora que você já sabe animar uma figura, podemos passar ao programa que faz com que ela se mova pelo vídeo:

```
10 FOR N=0 TO 27
20 PRINT AT 10,N;"ooo<";AT 9,
N;"(((";AT 11,N;"((("
30 PRINT AT 10,N;"    ";AT 9,
N;"   ";AT 11,N;"   "
40 PRINT AT 10,N;"ooo<";AT 9,
N;")))";AT 11,N;")))"
50 PRINT AT 10,N;"ooo<";AT 9,
N;"   ";AT 11,N;"   "
60 NEXT N
70 GOTO 10
```

Este programa também usa um laço **FOR...NEXT**, porém com uma finalidade completamente diferente da do exemplo acima. Lá o computador fazia uma contagem, por frações de segundo, antes de imprimir uma imagem. Agora ele é responsável pela movimentação da imagem, que assume uma posição de cada vez na superfície da tela.

E por que a linha 10 mostra "**O TO 27**", se a tela do Sinclair tem, no total, 32 posições? Para descobrir, modifique a linha 10 para:

```
10 FOR N=0 TO 32
```

Outra dúvida pode ser em relação às linhas 30 e 50. Experimente apagá-las e você logo descobrirá sua utilidade.

MSX

O programa que movimenta a "taturana" usa exatamente os mesmos caracteres que os outros computadores. O que muda é o comando para posicioná-lo, que nos computadores da linha MSX (por exemplo o HotBit, da Sharp) recebe o nome de **LOCATE**. Esse comando posiciona o cursor sobre a tela, em uma posição definida pelos dois números que se seguem ao comando **LOCATE**. O primeiro número refere-se à coluna, e o segundo, à linha. Assim, por exemplo, **LOCATE 12,4** coloca o cursor na coluna 12 e na linha 4. Qualquer comando **PRINT** que seja dado após isto escreverá o caractere (ou caracteres) a partir da posição definida pelo **LOCATE**. Agora, digite o programa:

```
5 CLS
10 LOCATE 18,10:PRINT "ooo"
20 LOCATE 18,9:PRINT ")))"
30 LOCATE 21,10:PRINT "<"
40 LOCATE 18,11:PRINT ")))"
42 FOR I=1 TO 30
47 NEXT I
50 LOCATE 18,10:PRINT "ooo"
60 LOCATE 18,9:PRINT "((("
70 LOCATE 21,10:PRINT "<"
80 LOCATE 18,11:PRINT "((("
82 FOR I=1 TO 30
87 NEXT I
90 GOTO 10
```

Quando o programa é executado, você pode observar a figura do "inseto"

vibrando rapidamente. Ela é criada pela sobreposição dos dois conjuntos de caracteres, que estão nas linhas com o comando **PRINT**. Ao mesmo tempo, o comando **GOTO**, na última linha, faz com que o programa se repita indefinidamente, pois retorna sempre ao seu início.

O movimento das patinhas da "taturana" é bastante veloz, e quase não dá para perceber o efeito da animação. Por isto, precisamos dar um jeito para diminuir a velocidade. A maneira mais fácil de se fazer isto é utilizarmos a declaração **FOR...NEXT**, que cria um laço de repetição 'no vazio', isto é, apenas para gastar um pouco de tempo (explicaremos na próxima lição de programação BASIC como se usa este tipo de laço).

Assim, introduza a linha:

```
45 FOR T=1 TO 100 : NEXT
```

A seguir, rode o programa e observe como o efeito de movimentação das patas tornou-se muito mais lento. Isso foi possível porque o laço criado na linha 45 funciona como um contador — neste caso, contando até 100, quando então, ao terminar a contagem, prosseguirá executando o programa a partir da linha 50.

Pode-se variar a duração dessa pausa simplesmente mudando-se o número 100 para outro qualquer. Quanto maior for esse valor numérico, maior será o retardo de tempo provocado.

IMAGENS EM MOVIMENTO

O próximo passo é alterar o programa de forma que o corpo do inseto se movimente pela tela. Para isto, utilizamos a declaração **LOCATE**, já descrita.

Neste caso, usaremos uma variável para fazer com que a função **LOCATE** mude continuamente a posição do cursor, de modo que os **PRINT**'s que produzem a imagem dêem a impressão de deslocamento. Para fazer variar o valor desta variável que afeta o **LOCATE**, usamos um outro laço **FOR...NEXT** mais externo àquele que anima o bichinho:

```
10 FOR N=0 TO 36
20 LOCATE N,15:PRINT "OOO<"
30 LOCATE N,14:PRINT "((("
40 LOCATE N,16:PRINT "((("
50 LOCATE N,15:PRINT "    "
60 LOCATE N,14:PRINT "    "
70 LOCATE N,16:PRINT "    "
80 LOCATE N,15:PRINT "OOO<"
90 LOCATE N,14:PRINT ")))"
100 LOCATE N,16:PRINT ")))"
110 LOCATE N,15:PRINT "    "
120 LOCATE N,14:PRINT "    "
130 LOCATE N,16:PRINT "    "
140 NEXT N
```

O que fizemos foi simplesmente trocar o comando **GOTO** na linha 90 do programa anterior pelos comandos **FOR...NEXT**, que aumentam de um em um o valor da variável **P**, cada vez que o programa se repete. Como **P** é argumento da instrução **LOCATE** (indexando o número de linha, ou o eixo X), isto faz a figura se movimentar um passo para a direita, a cada ciclo do programa.

As linhas 50, 60, 70, 110, 120 e 130 apagam as posições antigas do inseto. Retire-as e veja o que acontece.

Quando você roda o programa, o inseto atravessa a tela e pára em sua margem direita. Para fazê-lo voltar à origem e começar o "passeio" novamente, experimente digitar a seguinte linha adicional ao programa acima:

```
160 GOTO 10
```

COMO CHEGAR AOS CARACTERES GRÁFICOS

Os computadores da linha MSX tem um grande conjunto de caracteres gráficos à disposição do programador dotado de instinto exploratório. Eles são digitados a partir do próprio teclado, e podem ser usados para criar figuras e desenhos mais elaborados.

Para ter acesso a estes caracteres, você deve utilizar as teclas **GRAPH, CODE** e **SHIFT**, juntamente com as demais teclas do teclado. No manual do seu computador podem ser encontradas "mapas" do teclado, que indicam as te-

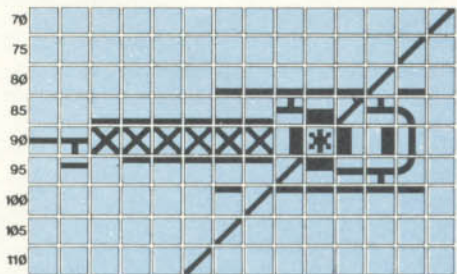
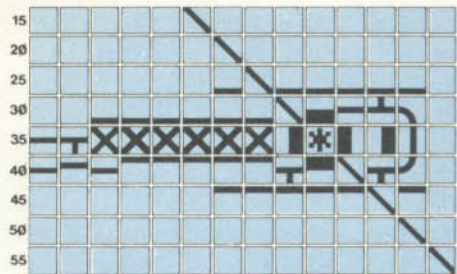
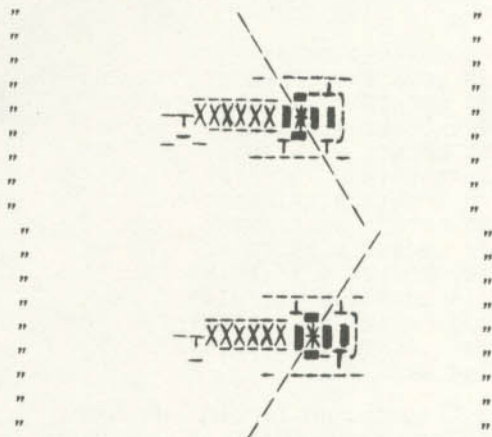
clas associadas a cada caractere gráfico existente.

Agora, tente digitar alguns caracteres gráficos. O símbolo de espadas do baralho, por exemplo, pode ser obtido pressionando-se as teclas **GRAPH** e **Ç** (cedilha). O símbolo do naipe de ouro pode ser obtido pressionando-se as teclas **SHIFT**, **GRAPH** e **Ç** simultaneamente.

O programa abaixo faz a animação gráfica de um helicóptero visto de cima, demonstrando como podem ser usados os caracteres gráficos dos MSX.

```
5 CLS
10 LOCATE 13,5:PRINT "                    "
20 LOCATE 13,6:PRINT "                    "
30 LOCATE 13,7:PRINT "                    "
40 LOCATE 13,8:PRINT "                    "
50 LOCATE 13,9:PRINT "                    "
60 LOCATE 13,10:PRINT "                   "
70 LOCATE 13,11:PRINT "                   "
80 LOCATE 13,12:PRINT "                   "
90 LOCATE 13,13:PRINT "                   "
100 LOCATE 13,5:PRINT "                   "
110 LOCATE 13,6:PRINT "                   "
120 LOCATE 13,7:PRINT "                   "
130 LOCATE 13,8:PRINT "                   "
140 LOCATE 13,9:PRINT "                   "
150 LOCATE 13,10:PRINT "                  "
160 LOCATE 13,11:PRINT "                  "
170 LOCATE 13,12:PRINT "                  "
180 LOCATE 13,13:PRINT "                  "
190 GOTO 10
```

Os possuidores de micros compatíveis com a linha Apple II e derivados, bem como os que têm um Microdigital TK-2000, podem rodar os mesmos programas listados acima para os micros da linha MSX. Algumas modificações, porém, devem ser feitas, face às diferenças dos modelos:

• todos os comandos **CLS** (usados para limpar a tela) devem ser substituídos pelo comando equivalente **HOME**.

Como construir um helicóptero.

• nos micros da linha Apple não há a instrução **LOCATE**, mas ela pode ser substituída por duas outras, que são **HTAB** e **VTAB** (tabulação horizontal e vertical, respectivamente).

Essas duas funções controlam o posicionamento horizontal e vertical do cursor, antes de dar um **PRINT**. O comando **HTAB** é acompanhado de um número entre 1 e 40, que determina a coluna de posicionamento; já o comando **VTAB** faz o mesmo com a linha (de 1 a 21).

Assim, por exemplo, onde está escrito, no programa para o MSX:

```
20 HTAB 18:VTAB 10:PRINT "OOO"
```

substitua por:

```
20 LOCATE 18,10:PRINT "OOO"
```

O programa do helicóptero não pode ser introduzido em micros da linha Apple, pois estes não possuem caracteres gráficos. O mesmo não acontece com o TK-2000, cujos caracteres estão disponíveis através do teclado, ao se pressionar as teclas **<CONTROL>** e **B**, simultaneamente, ou por meio da função **CHR$ (442)**, através de programa. Consulte o manual do TK-2000 e identifique os códigos dos caracteres usados nesse programa.

Se você quiser inventar novas figuras, basta usar os gráficos da tabela existente no manual ou os caracteres padronizados do teclado, ou os dois juntos. Caso seu computador seja diferente dos mencionados acima, proceda da seguinte maneira: desenhe as figuras em papel quadriculado e faça seu próprio programa, utilizando os caracteres gráficos disponíveis no seu computador.

NÚMEROS AO ACASO

Pense em um número ao acaso: é assim, aleatoriamente, que muitos jogos são criados pelo computador.

Ninguém aprende a jogar futebol praticando apenas uma jogada de cada vez, para só entrar numa partida quando todos os lances possíveis estiverem assimilados. Se alguém escolhesse esse método, teria de esperar a vida inteira, pois o número de jogadas possíveis é praticamente infinito.

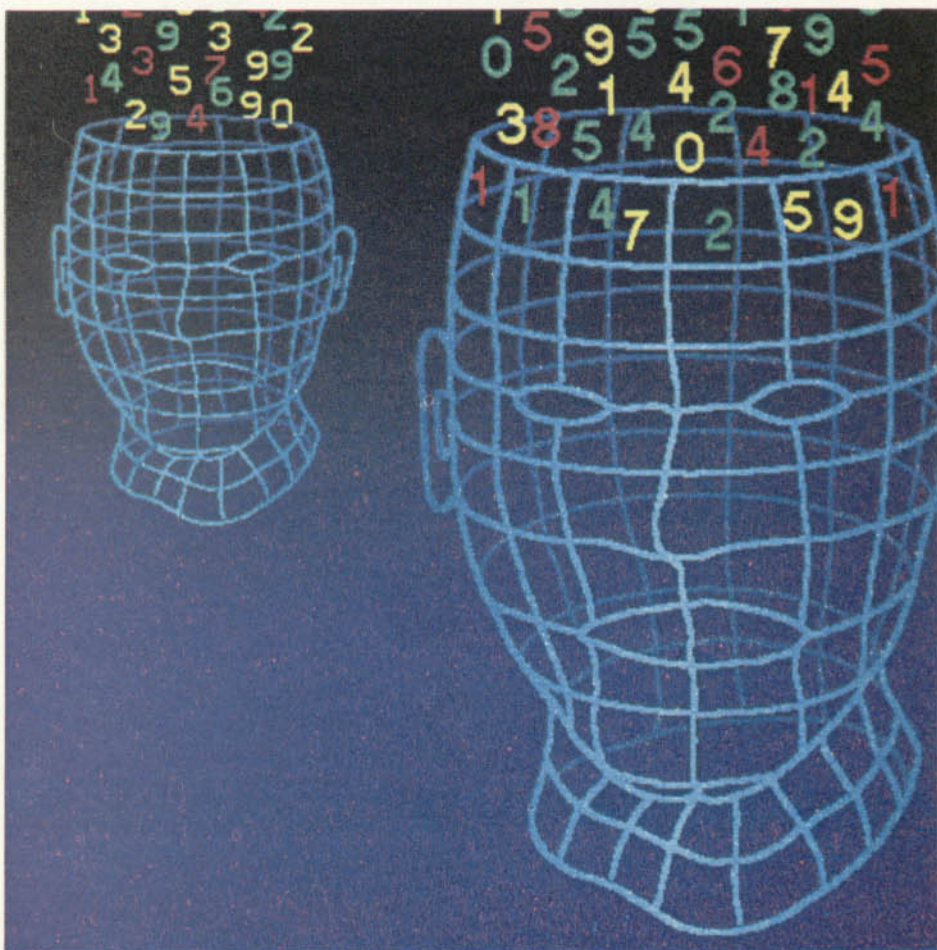
O mesmo, felizmente, não acontece com a programação de computadores, que pode ser aprendida de modo gradual. Assim, os manuais que acompanham os computadores ensinam uma função ou comando de cada vez. Você pode aprender por ele, se quiser. Mas a forma mais divertida de fazer isso é começar a jogar desde o começo, aprendendo na prática.

O jogo mais fácil de ser programado em um micro doméstico é aquele em que o computador "inventa" um número aleatório (ao acaso), e o jogador tenta adivinhá-lo.

A FUNÇÃO RND

Os computadores domésticos têm um gerador de números aleatórios (ou randômicos) que permite inventar jogos. Ele é operado, em BASIC, pela função **RND**. Em alguns computadores, entretanto, os números produzidos não são muito úteis na sua forma original — são todos frações decimais entre 0 e 0.99999999. Para verificar isso, digite este programa, teclando primeiramente **NEW** para limpar da memória qualquer programa já existente:

```
10 LET X=RND(0)
20 PRINT X
30 GOTO 10
```

```
10 LET X=RND
20 PRINT X
30 GOTO 10
```

```
10  LET X =  RND (1)
20  PRINT X
30  GOTO 10
```

(Lembre-se de teclar <**ENTER**> ou <**RETURN**> - dependendo do seu computador - após cada linha do programa.)

Quando você executar esse programa (use comando **RUN**), vai obter uma seqüência de longos números decimais, muito distante de um jogo de adivinhação (para saber por que é assim, veja o quadro *Perguntas e Respostas* na página 13).

Então, como conseguir que o computador produza somente números inteiros? A resposta é muito simples: adicionando a função **INT** (uma abreviação da palavra inglesa *integer*, que quer dizer inteiro) a estas linhas.

```
10 LET X=INT (RND*6)
```

```
10  LET X =  INT ( RND (1) * 6)
20  PRINT X
30  GOTO 10
```

Isso vai gerar números inteiros entre 0 e 5. Nos computadores da linha TRS-80, não é preciso utilizar a função **INT**, pois a função **RND** a realiza se receber um número inteiro positivo como argumento:

```
10 LET X=RND(6)-1
20 PRINT X
30 GOTO 10
```

(Lembre-se sempre de pressionar <**RETURN**> ou <**ENTER**> após digitar cada linha.)

Qualquer que seja o computador que estiver usando, você não está limitado a trabalhar com números entre 0 e 5, podendo escolher igualmente 10 ou 10 000 como número máximo: o computador sempre sorteará um valor compreendido entre 0 e o número máximo escolhido.

APRENDA A USAR VARIÁVEIS

Ao escrever o programa acima, além de selecionar um número aleatório, você deu um nome a ele (**X**). Daí em diante, no programa, toda vez que o **X** for colocado, o computador "saberá" que você se refere àquele número aleatório.

Esse nome, que permite ao computador identificar um valor, de tal forma que possa compará-lo com outro número, fazer operações aritméticas com ele, etc., é chamado de *variável*. A variável corresponde, grosso modo, ao rótulo de uma caixinha individual na memória do computador, onde armazenamos valores.

A DECLARAÇÃO INPUT

Em nosso joguinho, após ter gerado o número aleatório, o passo seguinte será avisar o computador de modo a fazê-lo aceitar seu palpite. Para isso, você deve utilizar a declaração **INPUT**. Ela diz ao computador para ficar esperando até que alguma informação tenha sido digitada pela pessoa que está usando o programa.

A declaração **INPUT** sozinha, entretanto, não tem sentido. Precisamos fornecer ao computador um nome de variável para que ele saiba como identificar o dado a ser digitado e armazenar o seu valor na memória. Vamos supor que queremos entrar um dado na variável **G** (de *guess*, ou palpite, em inglês). Poderia ser perfeitamente um outro nome qualquer, envolvendo uma ou mais letras combinadas, tal como **PALPITE**, se você achar interessante.

Assim, a declaração completa fica da maneira como segue (não a digite ainda):

```
INPUT G
```

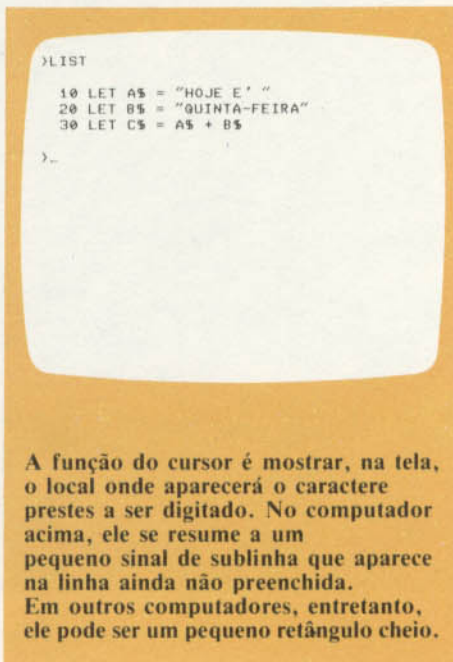
Agora que o computador tomou conhecimento do seu palpite, pode compará-lo com o número secreto gerado (que está guardado na variável **X**). Isso é feito de uma forma muito simples, equivalente à frase:

```
SE X=G ENTÃO ESCREVA"MUITO BEM!"
```

Em programação BASIC fica assim:

```
IF X=G THEN PRINT 'MUITO BEM!'
```

A declaração **IF ... THEN**, sem dúvida, é muito útil. Você vai usá-la muitas vezes na programação.

A função do cursor é mostrar, na tela, o local onde aparecerá o caractere prestes a ser digitado. No computador acima, ele se resume a um pequeno sinal de sublinha que aparece na linha ainda não preenchida. Em outros computadores, entretanto, ele pode ser um pequeno retângulo cheio.

Os micros das linhas TRS-80, TRS-Color e MSX têm uma extensão a essa declaração que é o **IF ... THEN ... ELSE** (em bom português, SE ... ENTÃO ... SENÃO). Neste caso, o programador pede para o computador fazer alguma coisa a mais, caso a condição de teste (por exemplo, se os números são iguais) não seja satisfeita. Nos computadores que não têm a declaração **ELSE** — como é o caso dos compatíveis com a linha Apple e com as linhas Sinclair ZX-81 e Spectrum — se os dois números não forem iguais o programa passará automaticamente para a linha seguinte àquela onde está o comando **IF**.

LISTAGEM DO PROGRAMA

Agora digite o programa abaixo (o sinal < > significa "é maior que e menor que" ou diferente de):

```
20 LET X=RND(6)-1
30 PRINT "O COMPUTADOR ESCOLHEU
 UM NUMERO ENTRE 0 E 5. VOCE PO
DE ADIVINHA-LO?"
40 INPUT G
60 IF G=X THEN PRINT "MUITO BEM
!"ELSE PRINT "QUE AZAR - VOCE E
RROU!"
```

Para os micros compatíveis com o ZX-81, digite em maiúsculas.

```
20 LET X=INT (RND*6)
30 PRINT "O Computador escolh
eu um numero entre 0 e 5. Ten
te adivinha-lo."
40 INPUT G
60 IF G=X  THEN  PRINT "Muito
bem!"
80 IF G<>X  THEN  PRINT "Que
azar - Voce errou!"
```

```
20 LET X=INT(RND(1)*6)
30 PRINT"O Computador escolheu
um numero entre  0 e 5. Tente a
divinhá-lo."
40 INPUT G
60 IF G=X THEN PRINT "Muito bem
!"
80 IF G<>X THEN PRINT "Que azar
 - Você errou!"
```

Rode este programa e você verá que já dá para jogar, mas ainda não é muito divertido. Para começar, a tela vai ficando um tanto congestionada e, o que é pior, o jogo acaba logo após a primeira tentativa!

Para resolver o primeiro problema, você só precisa acrescentar:

```
10 CLS
50 CLS
```

```
10  HOME
50  HOME
```

A declaração **CLS** quer dizer "limpe a tela" (*clear screen*). Para os computadores da linha Apple, a declaração

equivalente é **HOME**, que significa "vá para casa" (no caso, o cursor vai para o alto da tela quando esta fica limpa). Assim, somente as informações novas aparecem no vídeo. Lidar com o segundo problema é um pouco mais difícil. Uma forma de resolvê-lo seria colocar um **GOTO** na linha 90, de modo a reiniciar o jogo automaticamente. Mas isso faria com que a tela limpasse bruscamente, e o jogador não poderia ver a tempo a mensagem:

```
90 GOTO 10
```

VARIÁVEIS ALFANUMÉRICAS

Uma maneira melhor é oferecer ao jogador a opção de um novo jogo apenas no caso de ele o desejar. Pode parecer um pouco complicado, mas, na prática, é bastante simples.

Comece digitando o programa:

```
10 CLS
20 LET X=RND(6)-1
30 PRINT "O COMPUTADOR ESCOLHEU
 UM NUMERO ENTRE 0 E 5. VOCE PO
DE ADIVINHA-LO?"
40 INPUT G
50 CLS
60 IF G=X THEN PRINT "MUITO BEM
!"ELSE PRINT "QUE AZAR - VOCE E
RROU!"
90 PRINT "VOCE QUER TENTAR OUTR
A VEZ?":PRINT"SE QUISER,DIGITE
'S' E PRESSIONE<ENTER>"
100 INPUT A$
110 IF A$="S" THEN GOTO 10
120 GOTO 100
```

Para o programa abaixo rodar corretamente nos compatíveis com o ZX-81, digite tudo em maiúsculas e substitua a linha 110 por:

```
110 IF A$="S" THEN GOTO 10
```

```
10 CLS
20 LET X=INT (RND*6)
30 PRINT "O Computador escolh
eu um numero entre 0 e 5. Ten
te adivinha-lo."
40 INPUT G
50 CLS
60 IF G=X  THEN  PRINT "Muito
 bem!"
70 IF G=X THEN  GOTO 90
80 IF G<>X  THEN  PRINT "Que
azar - Voce errou!"
90 PRINT "Voce quer tentar ou
tra vez?    Se quiser, digit
e S e pressione ENTER"
100 INPUT A$
110 IF A$="S" OR A$="s" THEN
GOTO 10
120 GOTO 100
```

```
10  HOME
20  LET X =  INT ( RND (1) * 6)
30  PRINT "O Computador sorteou
 um numero entre 0 e 5. Sera qu
e  voce consegue advinhar ?"
40  INPUT G
50  HOME
60  IF G = X THEN  PRINT "Muito
 bem !"
70  IF G <  > X THEN  PRINT " Q
ue azar ! - Voce errou."
90  PRINT "Quer brincar de novo
 ? Se quiser, tecle 'S' e apert
e a tecla RETURN."
100  INPUT A$
110  IF A$ = "S" THEN  GOTO 10
120  GOTO 100
```

Você já deve ter-se perguntado que significado tem a pontuação nos programas de computadores. Vejamos: as linhas no topo da tela são as instruções do programador: as que vêm a seguir mostram os resultados de execução dessas instruções. Uma vírgula, por outro lado, significa "tabule"; já um ponto e vírgula quer dizer "justaponha"; e um apóstrofo no Spectrum significa "imprima na próxima linha".

```
10 CLS
20 LET X=INT(RND(1)*6)
30 PRINT"O Computador escolheu
um numero entre  0 e 5. Tente a
divinhá-lo."
40 INPUT G
50 CLS
60 IF G=X THEN PRINT "Muito bem
!"
70 IF G=X THEN GOTO 90
80 IF G<>X THEN PRINT "Que azar
 - Você errou!"
```

Como se especificam os intervalos de números aleatórios?

As funções **RND** nos micros da linha Sinclair, **RND(1)** no Apple **II**, TK-2000 e MSX, e **RND(0)** nos compatíveis com o TRS-80 e Color, geram números aleatórios distribuídos entre 0 e 0.9999999. Se você quiser um número aleatório distribuído em uma faixa maior de números, multiplique o número obtido através das formas acima por uma constante. Por exemplo, para obter um número entre 0 e 39.99999999, basta multiplicar a saída da função **RND** por 40.

Para gerar números inteiros, use a função **INT**. Um número inteiro entre 0 e 39 é produzido através da expressão **INT** (função **RND** original * 40). Se você quiser números distribuídos entre 1 e 40 adicione 1 à expressão anterior. Nos micros da linha TRS-80, para gerar um número aleatório entre 1 e 40, use **RND**(40).

Alguns computadores utilizam as palavras RAND, RANDOM ou RANDOMIZE. Para que servem?

Para evitar a repetição da seqüência de números aleatórios gerada pelo computador. Os micros da linha TRS-80 têm a declaração **RANDOM**, e os da linha ZX-81, a **RAND**. Nestes últimos, **RAND 1** (ou **RANDOMIZE 1**, no caso do Sinclair Spectrum e compatíveis) assegura que a seqüência aleatória seja constante. Quando usamos **RAND** ou **RANDOMIZE** sem argumento, ou seguidos de 0, a seqüência nunca se repete.

```
90 PRINT "Você quer tentar outr
a vez?           Se quiser, di
gite S e pressione RETURN"
100 INPUT A$
110 IF A$="S" OR A$="s" THEN GO
TO 10
120 GOTO 100
```

Como você pode notar, primeiro se pergunta ao jogador (linha 90) se ele quer jogar novamente. Depois, para avisar o computador que espere por uma resposta, usa-se a declaração **INPUT** na linha 100.

Mas, desta vez existe uma diferença importante. Depois da linha 20, o jogador deu entrada a um número. Agora ele vai entrar um **S** (para "sim") ou um **N** (para "não"), ou seja, uma letra não e um número.

Isso significa que, na linha 100, ao invés de **INPUT A** você deverá usar **INPUT A$**. O cifrão (também chamado de "dólar"), designa o chamado **string** (cordão) e **A$** é uma variável alfanumérica (*string variable*).

Por que o $ é necessário? Para entender isso é preciso conhecer a maneira como o computador armazena dados e trabalha com entradas, o que veremos mais adiante. Por enquanto, o importante é lembrar o seguinte: quando o computador deve esperar um número, você usa **INPUT A, INPUT B, INPUT X** ou qualquer outra letra; quando se trata de uma letra ou palavra, você deve usar **INPUT A$, INPUT B$, INPUT X$** etc.

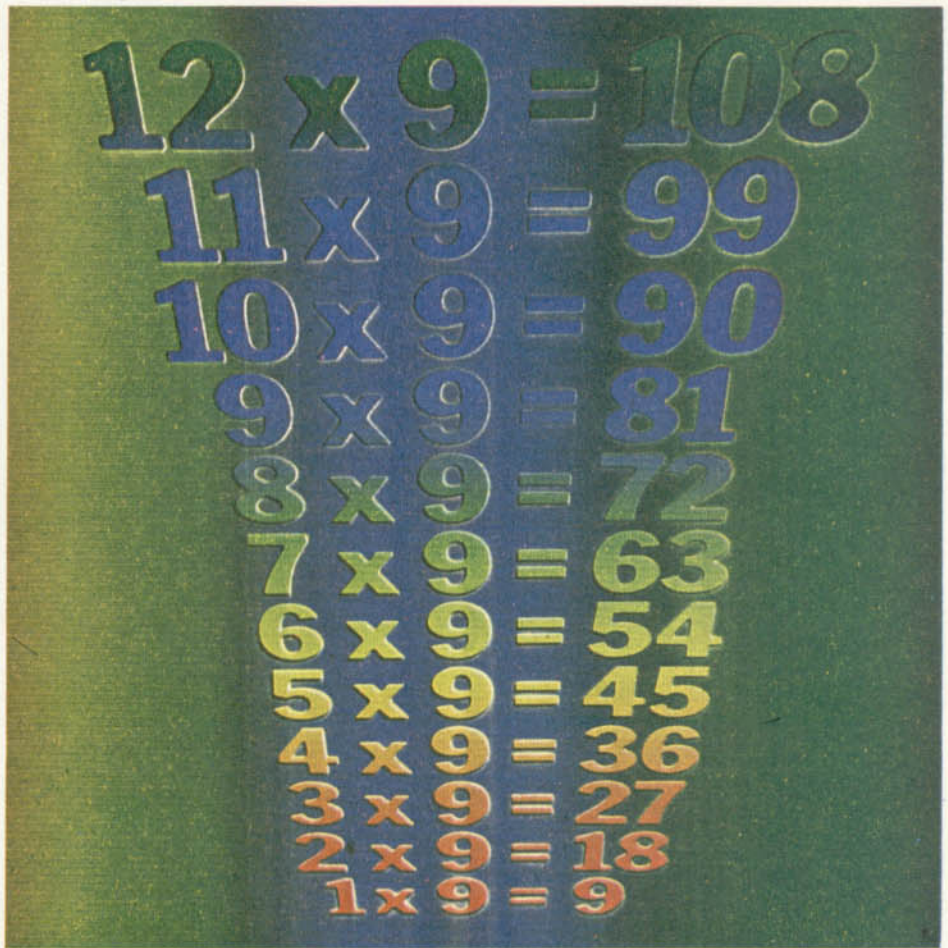
A linha 120 foi incluída de forma que, se o jogador não deseja outra partida imediatamente, o computador espera até que ele queira, repetindo o processo até que a resposta seja igual a **S**. Isto acontece devido ao retorno constante à linha 100 até que a tecla **S** seja pressionada, quebrando o ciclo.

VOCÊ SABE TABUADA?

A função **RND** tem centenas de usos em programação. Imagine, por exemplo, que você deseje ensinar ao seu filho ou irmão a tabuada do nove. Você poderia fazer o seguinte:

```
10 PRINT "QUANTO E 1 VEZES 9?"
20 INPUT A
30 IF A=9 THEN PRINT "CORRETO!"
40 PRINT "QUANTO E 2 VEZES 9?"
50 INPUT B
60 IF B=18 THEN PRINT "CORRETO!"
```

Mas, desse jeito, você teria um programa muito longo que não resolveria nem mesmo o problema de como proceder se uma das respostas estiver errada! Devemos usar a função **RND**, então, para produzir um programa que seja mais compacto e faça as perguntas corretamente, em seqüência aleatória. Outra recomendação importante: trabalhe primeiro na parte principal do programa, deixando os enfeites para depois. Assim, experimente agora estas linhas (lembre-se de digitar o comando **NEW** antes!):

```
10 LET N=RND(12)
20 PRINT "QUANTO E ";N;" VEZES
9?"
30 INPUT A
40 IF A=N*9 THEN PRINT "CORRETO
!"
```

Digite tudo o que vem abaixo apenas em maiúsculas, se for usar um compatível com ZX-81:

```
10 LET N=INT (RND*12+1)
20 PRINT "Quanto e ";N;" veze
s 9?"
30 INPUT A
40 IF A=N*9 THEN  PRINT "Corr
eto!"
```

```
10  LET N =  INT ( RND (1) * 12
 + 1)
20  PRINT "QUANTO E ";N;"VEZES
9?"
30  INPUT A
40  IF A = N * 9 THEN  PRINT "C
ERTO !"
```

```
10 LET N=INT(RND(-TIME)*12+1)
20 PRINT "Quanto é ";N;" vezes
9 ?"
30 INPUT A
40 IF A=N*9 THEN PRINT "Correto
!"
```

```
]PRINT (1 + 2) * 3
      9
]PRINT (4 + 5) / 3
      1.5
]PRINT (8 - 6) ^ 2
      4
]PRINT SQR (2)
1.41421356
```

Símbolos de computador utilizados em cálculos de aritmética elementar: o asterisco (*), e não o x, é empregado para indicar multiplicação; a barra (/) significa "dividido por"; a flechinha para cima indica "elevado à potência de..." (em alguns computadores é um sinal circunflexo; em outros, dois asteriscos **). Em compensação, os sinais que indicam as operações de soma e subtração são seus velhos conhecidos: o mais (+) e o menos (−) da aritmética elementar.

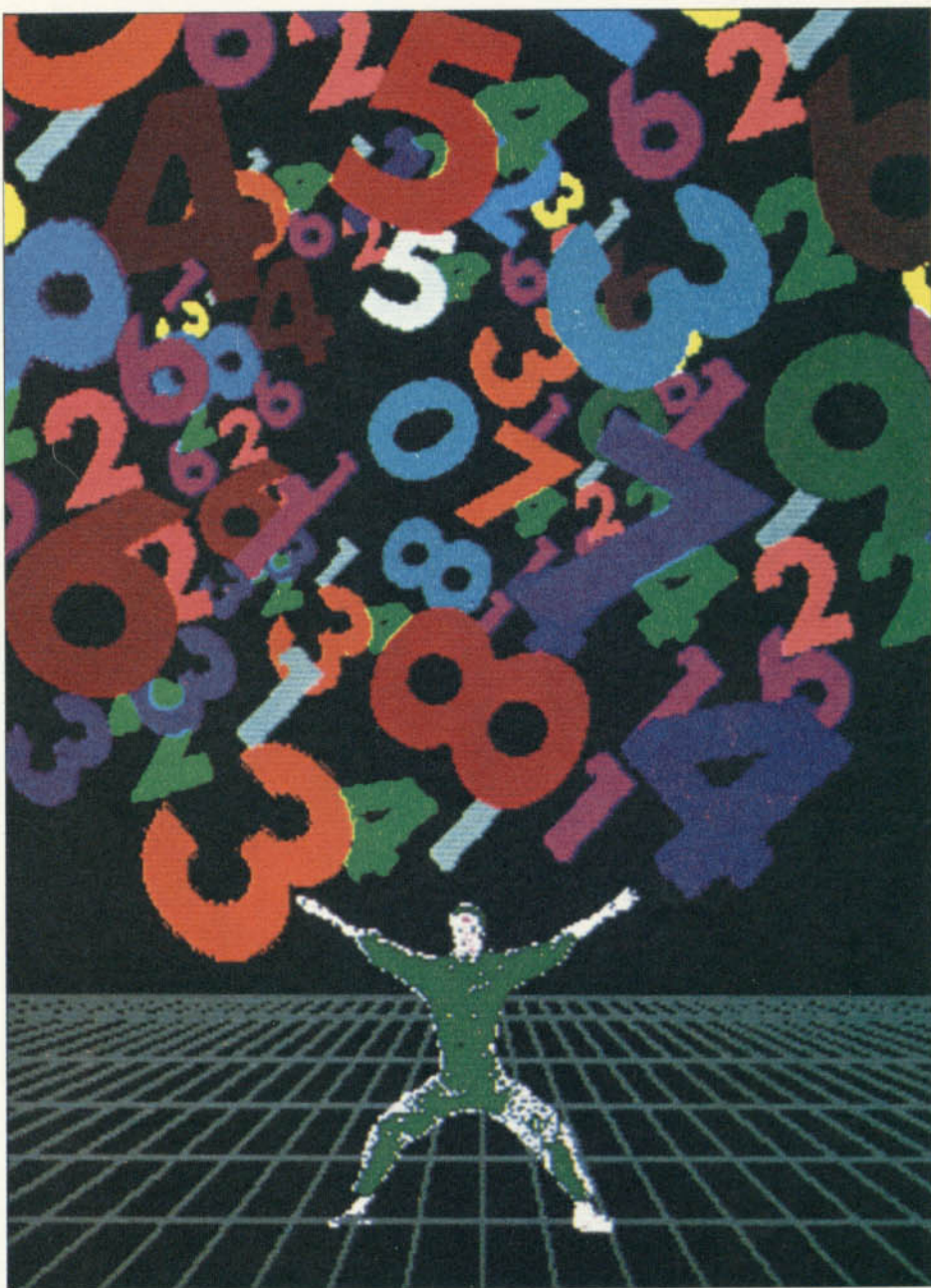
Note a função **TIME** sendo usada como argumento da função **RND**. Este é um artifício para gerar sempre números aleatórios diferentes nos micros da linha MSX, pois **TIME** informa ao computador o valor do tempo passado, em segundos, desde que o aparelho foi ligado.

Este programa usa o gerador **RND** de forma semelhante à do jogo de adivinhação. Na linha 10, determina-se uma variável para o número aleatório que o computador escolhe. Ela foi chamada de **N**, mas poderia ser outra letra ou palavra, desde que fosse usada sempre da mesma forma. Na parte direita da linha 10, o programa diz ao computador para escolher um número *inteiro* entre 1 e 12. Nos micros da linha Sinclair, é necessário somar 1, porque seus números aleatórios começam a partir de zero.

Na linha 20, pede-se ao jogador que multiplique por 9 o número que o computador escolheu desta vez. A linha 40 diz ao computador para multiplicar o número aleatório por 9 e comparar o resultado com a resposta dada pelo jogador (ou aluno). Se esta estiver correta, o computador mandará a mensagem de "CORRETO!" para a tela. Execute o programa e faça-o rodar várias vezes, para ver se funciona. Para que ele repita automaticamente as questões, acrescente a linha:

```
50 GOTO 10
```

Mas, pensando melhor, por que não fazemos as coisas de uma forma mais elegante, como se segue?

```
10 PRINT "OI. QUAL E O SEU NOME
?"
20 INPUT A$
30 CLS
40 PRINT "OLA, ";A$:PRINT"EU TE
NHO ALGUMAS PERGUNTAS PARA VOCE
"
50 FOR X=1 TO 3000:NEXT X
60 CLS
70 LET N=RND(12)
80 PRINT "QUANTO E ";N;" VEZES
9?"
90 INPUT A
100 IF A=N*9 THEN GOTO 150
110 CLS
120 PRINT A;"?"
130 PRINT "TENTE OUTRA VEZ."
140 GOTO 80
150 PRINT "MUITO BEM, ";A$;"!":
PRINT"AQUI VAI MAIS UMA:"
160 FOR X=1 TO 2000:NEXT X
170 GOTO 60
```

Digite tudo o que vem abaixo apenas em maiúsculas, se for usar um compatível com ZX-81:

```
10 PRINT "Oi. Qual e seu nome
?"
20 INPUT A$
30 CLS
40 PRINT "Ola, ";A$;".","Eu t
enho algumas perguntas para v
oce."
50 PAUSE 200
60 CLS
70 LET N=INT (RND*12)+1
80 PRINT "Quanto e ";N;" veze
s 9?"
90 INPUT A
100 IF A=N*9 THEN  GOTO 150
110 CLS
120 PRINT A;" ?"
130 PRINT "Tente outra vez!"
140 GOTO 80
150 PRINT "Muito bem, ";A$;"."
,"Aqui vai mais uma:"
160 PAUSE 150
170 GOTO 60
```

```
10   PRINT "Ola.Qual e o seu nom
e?"
20   INPUT A$
30   HOME
40   PRINT "Oi ";A$: PRINT "Eu t
```

```
enho algumas perguntas para voc
e."
50  FOR X = 1 TO 6000: NEXT X
60  HOME
70  LET N =  INT ( RND (1) * 10
 + 1)
80  PRINT "Quanto e ";N;" vezes
 9?"
90  INPUT A
100  IF A = N * 9 THEN  GOTO 15
0
110  HOME
120  PRINT A;"?"
130  PRINT "Errado. Tente outra
 vez"
140  GOTO 80
150  PRINT "Muito bem ";A$: PRI
NT "Aqui vai outra."
160  FOR X = 1 TO 4000: NEXT X
170  GOTO 60
```

```
10 PRINT "Oi. Qual é o seu nome
?"
20 INPUT A$
30 CLS
40 PRINT "Olá, ";A$;".","Eu ten
ho algumas perguntas para você:
"
50 FOR X=1 TO 2500:NEXT X
60 CLS
70 LET N=INT(RND(-TIME)*12)+1
80 PRINT "Quanto é ";N;" vezes
9?"
90 INPUT A
100 IF A=N*9 THEN GOTO 150
110 CLS
120 PRINT A;"?"
130 PRINT "Tente outra vez."
140 GOTO@80
150 PRINT "Muito bem, ";A$;"!",
"Aqui vai outra:"
160 FOR X=1 TO 1500:NEXT X
170 GOTO 60
```

As linhas 30, 60 e 110 evitam que a tela fique repleta de mensagens do computador ou de respostas erradas. As linhas 50 e 160 fazem o computador esperar algum tempo antes de formular a próxima pergunta. As declarações **FOR...NEXT** serão explicadas na próxima lição de BASIC.

No programa para o Sinclair, as vírgulas nas linhas 40 e 150 servem para espaçar as mensagens na tela. Nos outros programas, as declarações **PRINT** extras fazem o mesmo. Para evitar que o programa seja interminável, faça o seguinte:

```
>LIST

 10 LET A$ = "HOJE E' "
 20 LET B$ = "QUINTA-FEIRA"
 30 LET C$ = A$ + B$

>RUN
HOJE E' QUINTA-FEIRA
>NEW
>CLS
```

Os números colocados no começo de cada linha do programa são muito importantes. Sem eles, o computador executará cada linha diretamente e de forma separada, em vez de seguir o programa como um todo. Mesmo que elas sejam introduzidas fora de ordem, o computador as colocará na seqüência certa (ou seja, em ordem ascendente), orientando-se pelos números de linha. Intervalos entre os números são utilizados para permitir a introdução de novas linhas, caso isso seja necessário.

Pressione a tecla **BREAK**

Acione as teclas **STOP** e **ENTER**.

Pressione as teclas **CONTROL** e **C**, simultaneamente, e depois **RETURN**.

Pressione as teclas **CONTROL** e **STOP**, simultaneamente. Mudando os noves do programa para cincos, seis ou setes, você poderá testá-lo com outras tabuadas. Será que você conseguiria modificar o programa para colocar esses números como variáveis, que seriam definidas logo no começo por outra declaração **INPUT**?

Muitas vezes, os iniciantes encontram dificuldades para interromper um programa em execução e voltar para a listagem. Isso acontece especialmente durante uma série de **INPUTs**, quando o computador enche a tela de mensagens, não importa o que você digite.

A seguir, abordaremos alguns pequenos truques que o ajudarão a sair desses aparentes impasses. Não existem problemas sem solução para quem lida com computadores.

Pressione simultaneamente as teclas **CONTROL** e **SPACE** (para acionar a função **BREAK**). Se isso não der resultado, é porque o programa está parado numa declaração. **INPUT**. Neste caso, se houver aspas na parte de baixo da tela, use a tecla de recuo do cursor e **DELETE**, para remover as aspas à esquerda. Depois disso, e se não houver aspas, pressione as teclas **STOP** e **ENTER**, e acione **ENTER** de novo, para listar o programa automaticamente.

Você deve acionar a tecla **CTRL**, mantê-la pressionada e, a seguir, pressionar a tecla **C**. Usaremos a notação **CTRL-C** para essa operação, assim como em qualquer outra ocasião em que duas teclas devam ser acionadas simultaneamente **CTRL-C** não funcionar experimente **CTRL-RESET**. Após obter na tela o 'J', que é o sinal de prontidão, digite **LIST**.

Pressione a tecla **BREAK**, e depois digite o comando **LIST**. Se não funcionar, pressione a tecla **RESET** na parte de trás do computador.

Acione agora simultaneamente as teclas **CONTROL** e **STOP**.

Especificação de faixas de números aleatórios

Números aleatórios entre 0 e 0.9999999	RND (1)	RND (0)	RND
Números aleatórios entre 0 e N * 0.99999999	RND (1)*N	RND (0) *N	N*RND
Números aleatórios entre −10 e +10	INT(RND (1) *21)−10	RND (21)−11	INT(RND* 21)−10
Números aleatórios entre 0 e 39	INT(RND (1)* 40)	INT(RND (0)*40)	INT(RND* 40)
Números aleatórios entre 1 e 40	INT(RND(1)*40)+1	RND (40)	INT(RND*40)+1

ESCREVA CARTAS SEM ESFORÇO

- COMO ENTRAR O PROGRAMA
- APRENDA A ADAPTAR O PROGRAMA À SUA IMPRESSORA
- DIAGRAMAÇÃO DE UMA CARTA PADRÃO
- MELHORE SEU TEXTO
- IMPRESSÃO DE CÓPIAS

Você comete erros quando escreve cartas importantes? Se isso acontece, aqui está um programa que permite produzir cartas padronizadas com cópias personalizadas para muitas pessoas.

Escrever um grande número de cartas similares (por exemplo, pedidos de emprego, cartões de Natal, etc.) é uma tarefa longa e tediosa, quando realizada manualmente.

O programa apresentado neste artigo resolve num instante esse tipo de problema. Ele é um modelo simplificado de um editor de textos, que permite entrar com uma carta padronizada no computador e produzir, caso necessário, cópias ligeiramente diferentes entre si. Alguns computadores pessoais, como os da linha MSX, por exemplo, têm recursos para acentuação correta de textos em português, letras minúsculas e maiúsculas no teclado e na tela, etc. Já os computadores compatíveis com a linha Sinclair ZX-81 são bastante difíceis de usar com processamento de textos, pois dispõem de pouquíssimos recursos para isto. Assim, não apresentamos aqui uma versão para máquinas desse tipo.

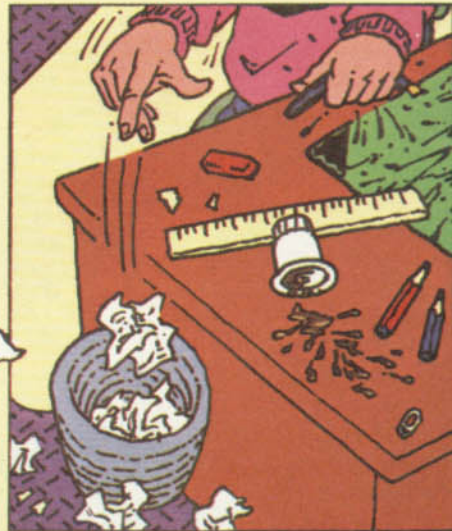
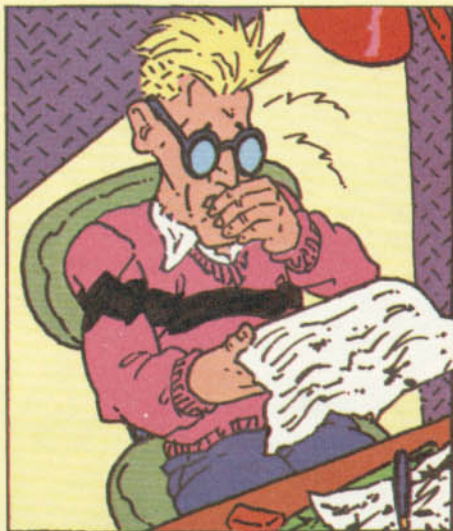
Ao contrário de um programa completo de processamento de textos, que costuma ser muito complexo e tomaria horas de digitação para ser colocado no computador, o programa aqui apresentado tem bem menos recursos. Por exemplo, não dá para ver na tela o aspecto final da carta a ser impressa. Mas não se preocupe, ele não é tão limitado assim: tem recursos para evitar a divisão de palavras ao final de uma linha, inserir uma linha em branco entre dois parágrafos, etc.

Qualquer tipo de impressora serve para produzir as cartas (lembre-se, entretanto, que se você usou acentuação no texto, é necessário que a sua impressora seja capaz de reproduzi-la, o que nem sempre acontece). Se você quiser uma melhor qualidade de impressão, como é o caso de cartas mais "profissionais", vai precisar, evidentemente, de uma impressora mais sofisticada (e com fita nova!). Esta pode ser do tipo matricial, com capacidade de imprimir em "qualidade carta" (dupla sensibilidade de pontos), ou uma impressora (ou máquina de escrever) acoplada ao computador, usando o sistema de margarida, ou outro. Essas máquinas são caras, e nem todo mundo pode tê-las. Assim, talvez você possa usar a de um amigo ou de seu trabalho. As características das impressoras para micros serão discutidas em um artigo posterior.

COMO "ENTRAR" O PROGRAMA

O programa consiste de duas partes: o programa propriamente dito, na primeira parte da listagem, e o texto da carta, que deve ser armazenado no programa, através de uma série de declarações **DATA**.

Comece digitando o programa principal, listado a seguir para cada tipo de máquina. Não digite as declarações **DATA** presentes no programa. Em seguida, grave o programa assim digitado em fita cassete ou disquete, usando o comando **SAVE** (em alguns computadores, esse comando chama-se **CSAVE**): consulte o manual do computador para ver como usá-lo. Assim, o programa gravado poderá ser usado para qualquer tipo de carta, depois. Se você quiser preservar em a fita ou disco o programa contendo algum mo-

delo particular de carta, grave-o separadamente, com outro nome (programa mais dados).

Antes de imprimir sua carta, você deve fazer alguns ajustes, dependendo da impressora utilizada.

Em primeiro lugar, verifique quantos caracteres por linha sua impressora aceita (geralmente são 40, 80, 120 caracteres por linha). Então, modifique o programa de forma a seguir essa largura de linha, permitindo uma margem adequada de ambos os lados. No programa, a variável **TL** representa o comprimento total disponível na impressora, e **LL**, o comprimento da linha a ser digitada. Assim, você pode alterar os valores atribuídos no programa a essas variáveis, antes de imprimir as cartas.

Modifique a linha 20 (dando a **TL** o valor da largura de linha disponível na impressora) e a linha 30, de modo que o valor de **LL** contenha a largura da linha da carta. Do jeito que está no programa, a linha 40 mostra na tela como sairá a carta, antes de imprimi-la. No momento da impressão, modifique a variável **P**, nessa linha, de 0 para 2.

Modifique a linha 10 de maneira que **PL** seja a largura da impressora, e **LL**, a largura que você deseja para a carta (no exemplo dado, essas larguras estão muito pequenas para impressoras mais profissionais).

Nesse computador não dá para imprimir a carta primeiro no vídeo, antes de mandá-la para a impressora, a não ser que você modifique todas as declarações tipo **LPRINT** existentes, para **PRINT**, e vice-versa. Na linha 10, as variáveis **PL** e **LL** devem ser modificadas conforme a sua impressora (ou, para impressão em vídeo, para **LL=35** e **PL=40**).

Modifique a linha 30 de maneira que **TL** e **LL** contenham as larguras de linha disponíveis na impressora e na carta, respectivamente. Se você quiser ver na tela, primeiro, como sairá a carta, mude o valor de **P** para 0, **TL** e **LL** para 39.

DIGITE SUA PRIMEIRA CARTA

Sua carta consistirá de uma série de declarações **DATA**, uma linha de texto por declaração. Digite-as como mostrado no exemplo na última página deste artigo.

As instruções **DATA** devem começar na linha 1000 do programa, e conter inicialmente o seu endereço. A primeira tarefa do programa será procurar a linha mais longa do endereço, e imprimi-lo à direita do cabeçalho. Depois disso, as declarações **DATA** deverão conter o texto propriamente dito da carta. O computador alinhará esse texto junto à margem esquerda da carta, conforme os padrões mais modernos para cartas comerciais.

Cada linha da carta deve começar com aspas, logo após a declaração **DATA**. Alguns *símbolos de edição* podem ser incluídos no texto. Eles têm por objetivo orientar a forma estética de impressão. Aqui vai o significado de cada símbolo de edição para nosso programa:
Aqui vai o significado de cada símbolo de edição para nosso programa:

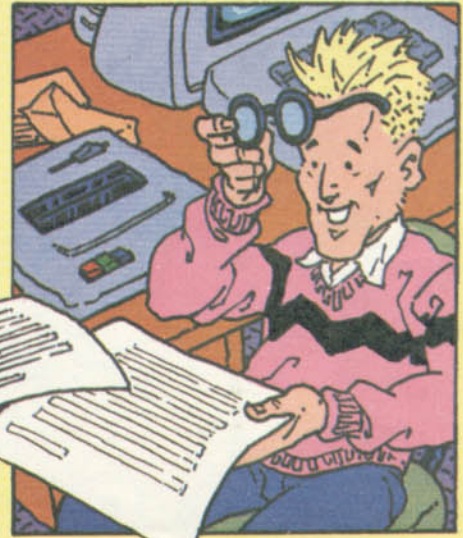
- O símbolo # significa: "esta linha é parte do meu endereço, coloque-a do lado direito da página".

- O cifrão, $, quer dizer: "comece um novo parágrafo, deixando uma linha em branco acima dele".

- O & comercial (*ampersand*): "comece uma nova linha na margem esquerda, mas não deixe uma linha em branco antes" (ele é útil para compor o endereço do destinatário da carta).

- O asterisco (*) significa: "centre esta linha".

Se você deseja colocar algumas linhas em branco indique-as com uma série de cifrões entre aspas, um para cada linha em branco. Quando você atingir o final da carta, os computadores do tipo abaixo necessitam de uma linha adicional:

Entre um número de linha adicional, seguido por **DATA $**
Entre outra linha adicional, seguida por **DATA**.

MELHORANDO SUA CARTA

Visto que tudo está escrito em BASIC, você poderá, se quiser, gravar novamente o programa, com as declarações **DATA** incluídas.

Posteriormente, se você quiser melhorar a carta, ou produzir uma nova versão, o primeiro passo será carregar o programa correspondente, a partir do disco ou cassete (usando o comando **LOAD**, ou **CLOAD**, conforme o tipo de computador), e editar o programa usando o editor BASIC da forma habitual. Grave-a novamente, se quiser preservar essas mudanças. Caso contrário, terá que redigitá-las quando carregar o programa original novamente.

IMPRIMINDO AS CARTAS

Para imprimir sua carta basta executar o programa: ele se encarregará de ativar e desativar a impressora. Se você tentar fazer isso sem que a impressora esteja conectada (quando ela está ligada, a luz que indica a conexão com o computador, normalmente rotulada **LINHA** ou **LINE**, permanece acesa), o computador ficará parado, esperando que a impressora entre em funcionamento. Neste caso, interrompa a execução do programa, pressionando a tecla **BREAK** ou sua equivalente. A forma de ver a carta no vídeo, antes de imprimi-la, já foi indicada.

O tipo de papel a ser usado depende de sua impressora. Algumas aceitam folhas soltas, caso tenham o rolo compressor de borracha: essas folhas são melhores para uma correspondência mais formal. A maioria das impressoras, entretanto, usa formulário contínuo, deslocado pelos picotes na margem (remalina). Depois de impressa a carta, retire as remalinas com uma guilhotina ou abrindo os picotes, acertando com uma tesoura ou lâmina afiada.

No caso de utilizar papel timbrado, não é necessário incluir o seu endereço na carta. Assim, digite o texto normalmente, começando da linha 1000, mas sem entrar as linhas que começam por # . Para imprimir cópias repetidas da mesma carta, execute o programa novamente, tantas vezes quantas forem necessárias.

T

```
10 CLEAR 200
20 TL=80
30 LL=56
40 P=0
50 SP=(TL-LL)/2:HW=TL/2
60 PRINT #-P,CHR$(13)
70 IFP=2 THEN SP$=STRING$(SP,"
") ELSE SP=0:HW=16
200 READA$:A$=A$+" "
210 IF A$=""GOTO 290
220 O$=LEFT$(A$,1)
230 IF O$="#" THEN ML=0:GOSUB 4
00:GOTO 290
240 IF O$="$" THEN RL=0:PRINT#-
P,CHR$(13);CHR$(13);SP$;:ST=2:G
OTO 280
250 IF O$="*" GOSUB800:RL=HW-LE
N(A$)/2:GOTO 280
260 IF O$="&" THEN RL=0:PRINT#-
P,CHR$(13);SP$;:ST=2:GOTO 280
270 ST=1
280 GOSUB 600
290 GOTO 200
400 IF LEN(A$)>ML THEN ML=LEN(A
$)
410 N=N+1:READ A$:IF LEFT$(A$,1
)="#" GOTO 400
420 IF ML>LL THEN CLS:PRINT "ER
RO NO FORMATO. ENDERECO MUITO L
ONGO PARA O TAMANHO  DE  LINHA
ESCOLHIDO.":END
430 RESTORE: FOR J=0 TO N-1:REA
D A$:PRINT#-P,STRING$(LL-ML," "
);SP$;:PRINT#-P,MID(A$-2);CHR$(
13);
440 NEXT:RETURN
600 WL=0
610 IF ST+WL>LEN(A$) THEN 630
620 IF MID$(A$,ST+WL,1)<>" " TH
EN WL=WL+1:GOTO 610
630 IF WL>LL THEN CLS:PRINT "ER
RO NO FORMATO...";A$:PRINT" ...
CONTEM UMA PALAVRA GRANDE DEMAI
S!":END
640 IF RL+WL-1>LL THEN PRINT #-
P,CHR$(13);SP$;:RL=0
650 WL=WL+1
660 PRINT #-P,MID$(A$,ST,WL);:R
L=RL+LEN(MID$(A$,ST,WL))
670 ST=ST+WL:IF ST<LEN(A$)+1 GO
TO 600
680 RETURN
800 IF LEN(A$)>LL THEN CLS:PRIN
T"ERRO NO FORMATO.  NAO  POSSO
    CENTRALIZAR",A$:END
810 PRINT #-P,CHR$(13);
820 IF HW>LEN(A$)/2 THEN PRINT
#-P,STRING$(HW-LEN(A$)/2," ");
830 ST=2:RETURN
```

S

```
10 LET LL=32: LET PL=32
15 LET LL=LL+1: LET T=(PL-LL)
/2
20 LET D=0
30 READ A$: LET L=LEN A$
40 LET C=0
50 IF C=L THEN  GOTO 30
60 LET C=C+1: LET D=D+1: IF C
>1 THEN  GOTO 100
70 IF A$(C)="#" THEN  GOTO
500
80 IF A$(C)="*" THEN  GOTO
700
85 IF A$(C)="&" THEN  GOTO
850
90 IF A$(C)="$" THEN  LPRINT
CHR$ 13;CHR$ 13;: LET D=0:
GOTO 900
95 LET A$=" "+A$: LET L=L+1
100 IF A$(C)=" " THEN  GOTO
800
110 LPRINT A$(C)
115 IF D>LL THEN  LET D=0
120 GOTO 50
500 LET NL=0: LET TA=LL: LET
BE=0
510 LET LE=LEN A$-1: IF LE>LL
THEN  PRINT  FLASH 1;"Erro no
formato - Endereco muito gran
de.": STOP
520 IF LE>BE THEN  LET BE=LE
530 LET NL=NL+1: READ A$: IF
A$(1)="#" THEN  GOTO 510
540 RESTORE 1000
550 LET TR=T+LL-BE: FOR G=1 TO
NL: FOR H=1 TO TR: LPRINT " ";
: NEXT H: READ A$: LPRINT A$(2
 TO ): NEXT G
560 GOTO 30
700 LET TA=(LL-L)/2+T: IF TA<T
THEN  PRINT CHR$ 13: PRINT
FLASH 1;"Erro no formato - Nao
posso cen-tralizar.": STOP
710 LPRINT CHR$ 13;: FOR n=1
TO ta: LPRINT " ";: NEXT n:
LPRINT A$(2 TO L): GOTO 20
800 LET SL=LL-D-1: LET CC=C+1:
LET X=1
810 IF CC=L THEN  GOTO 825
820 IF A$(CC)<>" " THEN  LET
CC=CC+1: LET X=X+1: GOTO 810
825 IF X>=LL THEN  PRINT CHR$
13: PRINT  FLASH 1;"Erro no Fo
rmato - Palavra muito grande."
: STOP
830 IF SL>=X THEN  GOTO 110
850 LPRINT CHR$ 13;: LET D=0
900 FOR B=1 TO T: LPRINT " ";:
NEXT B: GOTO 50
```

```
5 CLS
10 LET LL=35:LET PL=40
15 LET LL=LL+1:LET PL=(PL-LL)/2
20 LET D=0
30 READ A$:LET L=LEN(A$)
40 LET C=0
50 IF C=L THEN GOTO 30
60 LET C=C+1:LET D=D+1:IF C>1 T
HEN GOTO 100
65 B$=MID$(A$,C,1)
66 IF B$="=" THEN STOP
70 IF B$="#" THEN GOTO 500
80 IF B$="*" THEN GOTO 700
85 IF B$="&" THEN GOTO 850
90 IF B$="$" THEN LPRINT CHR$(1
3):LPRINT CHR$(13):LET D=0:GOTO
 900
95 LET A$=" "+A$:LET L=L+1
100 IF MID$(A$,C,1)=" " THEN GO
TO 800
110 LPRINT MID$(A$,C,1);
115 IF D>LL THEN LET D=0
120  GOTO 50
500 LET NL=0:LET TA=LL:LET BE=0
510 LET LE=LEN(A$)-1:IF LE>LL T
```

```
HEN PRINT "ERRO NO FORMATO - En
dereço muito longo"
520 IF LE>BE THEN LET BE=LE
530 LET NL=NL+1:READ A$:IF MID$
(A$,C,1)="#" THEN GOTO 510
540 RESTORE 1000
550 LET TR=T+LL-BE:FOR G=1 TO N
L:FOR H=1 TO TR:LPRINT " ";:NEX
T H:READ A$:F=LEN(A$)-1:LPRINT
RIGHT$(A$,F):NEXT G
560 GOTO 30
700 LET TA=(LL-L)/2+T:IF TA<T T
HEN LPRINT CHR$(13):PRINT "ERRO
 NO FORMATO - Não posso central
izar":STOP
710 LPRINT CHR$(13):FOR N=1 TO
TA:LPRINT " ";:NEXT N:F=LEN(A$)
-1:LPRINT RIGHT$(A$,F);:GOTO 20
800 LET SL=LL-D-1:LET CC=C+1:LE
T X=1
810 IF CC=L THEN GOTO 825
820 IF MID$(A$,CC,1)<>" " THEN
LET CC=CC+1:LET X=X+1:GOTO 810
825 IF X>=LL THEN LPRINT CHR$(1
3):PRINT "ERRO NO FORMATO - Pal
avra muito grande":STOP
830 IF SL>=X THEN GOTO 110
850 LPRINT CHR$(13):LET D=0
900 FOR B=1 TO T:PRINT" ";:NEXT
 B:GOTO 50
```

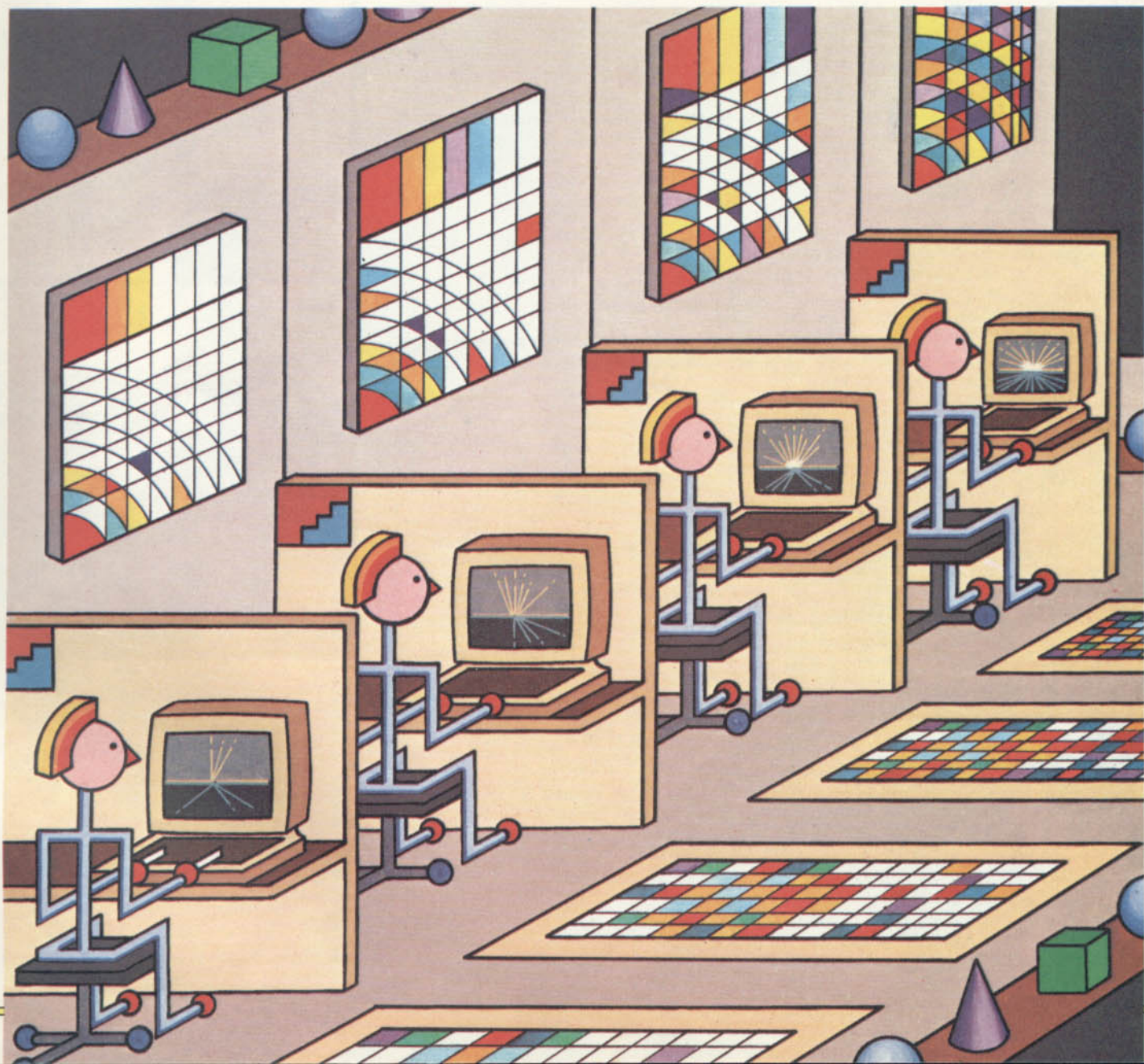
```
100  HOME
110  ONERR  GOTO 430
120 TL = 80:LL = 60:P = 1:D$ =
 CHR$ (4)
130 SP = (TL - LL) / 2:HW = TL
/ 2
140  PRINT D$;"PR#";P
150  PRINT  CHR$ (13)
160  READ A$:A$ = A$ +  CHR$ (3
2)
170  IF A$ = "" THEN 160
180 O$ =  LEFT$ (A$,1)
190  IF O$ = "#" THEN ML = 0: G
OSUB 260: GOTO 250
200  IF O$ = "$" THEN RL = 0: P
RINT  CHR$ (13); CHR$ (13); SPC
( SP);:ST = 2: GOTO 240
210  IF O$ = "*" THEN  GOSUB 40
0:RL = HW -  LEN (A$) / 2: GOTO
 240
220  IF O$ = "&" THEN RL = 0: P
RINT  CHR$ (13); SPC( SP);:ST =
 2: GOTO 240
230 ST = 1
240  GOSUB 310
250  GOTO 160
260  IF  LEN (A$) > ML THEN ML
=  LEN (A$)
270 N = N + 1: READ A$: IF  LEF
T$ (A$,1) = "#" THEN 260
280  IF ML > LL THEN  PRINT D$;
"PR#0": HOME : PRINT : PRINT "E
RRO DE FORMATO! O ENDERECO": PR
INT "E MUITO LONGO PARA O TAMAN
HO DA LINHA": END
290  RESTORE : FOR J = 1 TO N:
READ A$: PRINT  SPC( LL - ML +
SP + 1); RIGHT$ (A$, LEN (A$) -
 1); CHR$ (13);
300  NEXT : RETURN
310 WL = 0
320  IF ST + WL >  LEN (A$) THE
N 340
330  IF  MID$ (A$,ST + WL,1) <
 > " " THEN WL = WL + 1: GOTO 3
20
340  IF WL > LL THEN  PRINT D$;
"PR#0": HOME : PRINT "ERRO DE F
ORMATO! ": PRINT A$;"... CONTEM
 UMA PALAVRA MAIOR QUE A LINHA!
": END
350  IF RL + WL - 1 > LL THEN
PRINT  CHR$ (13); SPC( SP);:RL
= 0
360 WL = WL + 1
370  PRINT  MID$ (A$,ST,WL);:RL
 = RL +  LEN ( MID$ (A$,ST,WL))
380 ST = ST + WL: IF ST <  LEN
(A$) + 1 THEN 310
390  RETURN
400  IF  LEN (A$) > LL THEN  PR
INT D$;"PR#0": HOME : PRINT "ER
RO DE FORMATO! NAO SE PODE CENT
RALIZAR ";A$: END
410  PRINT  CHR$ (13);
420  IF HW >  LEN (A$) / 2 THEN
 PRINT  SPC( HW -  LEN (A$) /
2):ST = 2: RETURN
430  PRINT : PRINT D$;"PR#0": E
ND
```

A ARTE DE FAZER LAÇOS

- APRENDA A CRIAR EFEITOS SONOROS
- DESENHE NÚMEROS COM LAÇOS FOR... NEXT
- O NASCER E O PÔR-DO-SOL NO HORIZONTE DO VÍDEO
- CALEIDOSCÓPIOS

Os comandos FOR e NEXT servem para provocar repetições de um trecho de programa em BASIC. Muito versáteis, eles são "pau para toda obra" e podem ser empregados em quase tudo, desde jogos até programas comerciais.

Um laço é uma estrutura de programação usada em operações repetitivas. Ele é usado quando se quer que o computador conte até um certo número, executando, ao mesmo tempo, alguma outra operação. Além disso, o laço pode ser aplicado na repetição do mesmo padrão gráfico em diferentes posições na tela. A combinação de instruções **FOR** e **NEXT** na linguagem BASIC é usada na programação de diversos tipos de laços.

Assim, você aprenderá a criar "pinturas instantâneas" usando laços **FOR... NEXT** em programas muito pequenos. Esses laços serão igualmente úteis na programação de jogos e em programas de todos os tipos.

O QUE É UM LAÇO FOR... NEXT

Um laço **FOR... NEXT** em BASIC é um mecanismo que faz com que o computador repita uma mesma operação certo número de vezes.

Suponha, por exemplo, que você quisesse saber as raízes quadradas de todos os números de 1 a 100. Você poderia dizer ao computador:

```
PRINT SQR(1)
PRINT SQR(2)
PRINT SQR(3)
```

... e assim por diante.

E a cada pergunta o computador daria a resposta. Mas, além de impor um enorme trabalho de digitação aos seus dedos, esta não é uma técnica particularmente eficiente de uso do computador. Tente assim:

```
10 FOR N=1 TO 100
20 PRINT N,SQR(N)
30 NEXT N
40 PRINT "E ACABOU."
```

Esse programa faz com que o computador imprima 1 e sua raiz quadrada, 2 e sua raiz quadrada, 3 e sua raiz quadrada... e assim sucessivamente até atingir 100, no momento em que o aparelho pára.

Como ele faz isto? Quando o computador encontra um **FOR**, "sabe" que as linhas seguintes serão repetidas. Assim, ele as executa até encontrar o **NEXT**, volta à linha da instrução **FOR** e repete o processo.

Enquanto trabalha, o computador também está contando. A primeira vez em que passa pela linha 20 ele calcula a raiz quadrada de 1, na segunda, a de 2, e assim por diante.

Quando tiver trabalhado com o maior número da instrução **FOR**, ele deixará automaticamente o laço e continuará a execução do programa pela instrução seguinte ao **NEXT** — no nosso caso o **PRINT** da linha 40.

FRAÇÕES TAMBÉM

Ao executar um laço **FOR... NEXT**, o computador pode contar em unidades diferentes de 1. Para isso, usamos a declaração **STEP** (isto é "passo" ou "degrau") junto com a declaração **FOR**. Veja o exemplo:

```
10 FOR N=1 TO 30 STEP 2.7
20 PRINT N,SQR(N)
30 NEXT N
```

O computador não se atrapalha com o fato de que 30 não se divide em um número exato de passos, como você pediu. Ele vai o mais perto que pode e então pára.

O número de linhas entre o **FOR** e o **NEXT** tampouco o preocupa. Você pode colocar o **FOR** na linha 10 e o **NEXT** na 90 — ou mesmo 9000 — que ele não se esquecerá delas. Lembre-se, porém, de que o computador executará as linhas do laço a cada passagem.

COMO RETARDAR A AÇÃO

O laço **FOR... NEXT** tem dezenas de usos em programação. A mais simples delas é gastar tempo.

Se você rever a seção *Você sabe tabuada?* na primeira lição de BASIC, encontrará um bom exemplo. Nesse caso, tudo o que acontece entre cada **FOR** e o seu **NEXT** é que o computador "conta" um número. Essa contagem é muito rápida (o tempo exato pode ser da ordem de centésimos ou milésimos de segundo). Mas, enquanto espera que ele conte até 1000, você terá uma pausa bem perceptível. E se você executar:

```
10 FOR N=1 TO 1000000
20 NEXT N
```

... terá tempo, provavelmente, para tomar um café antes que ele termine.

APELO SONORO

Programadores de jogos freqüentemente tornam essas pausas menos cansativas, inserindo nelas algumas notas musicais. Tente este exemplo:

```
10 FOR I=255 TO 155 STEP -1
20 SOUND I,1
30 NEXT I
```

```
10 FOR n=29 TO 10 STEP -1
20 SOUND .015,n
30 NEXT n
```

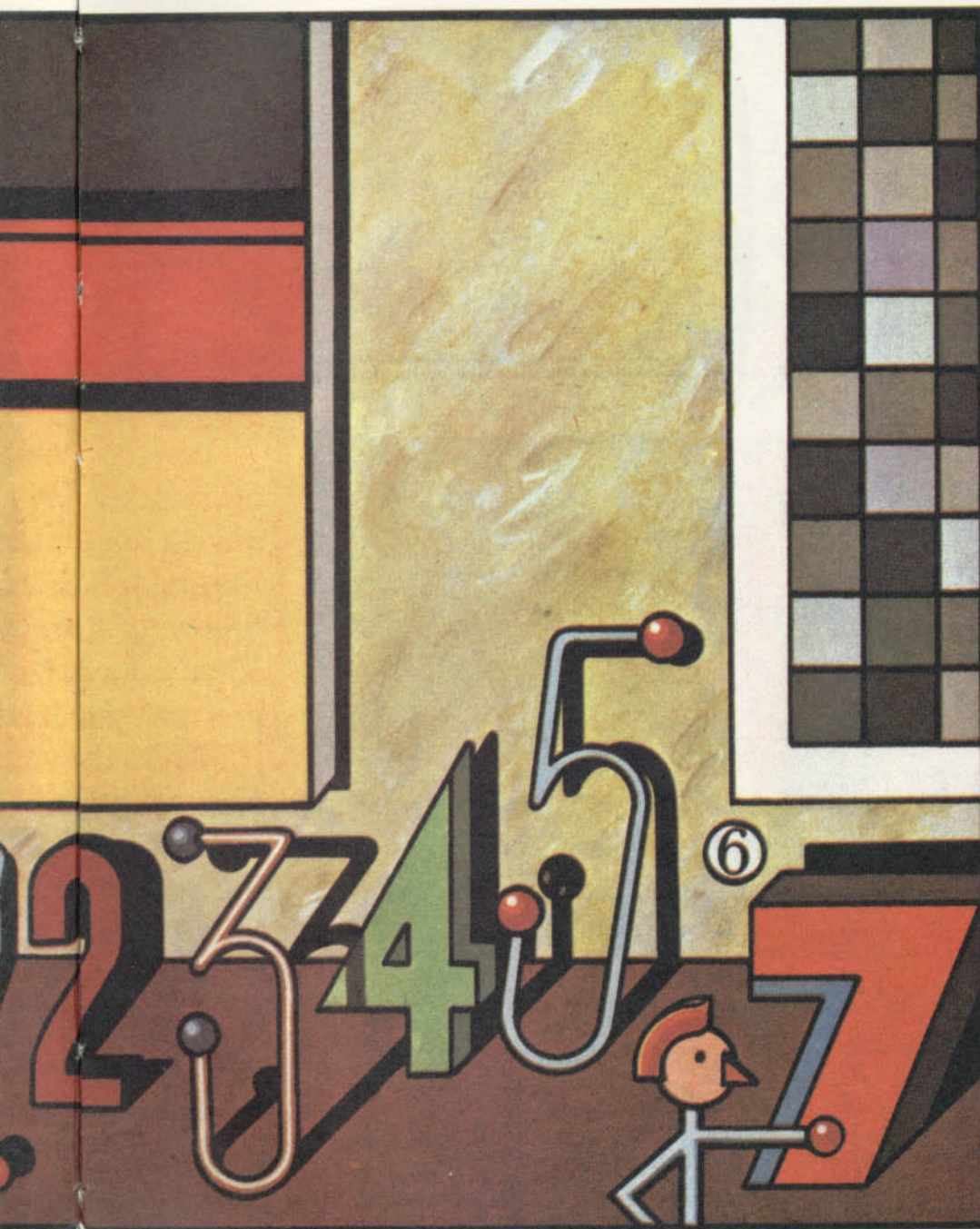
```
5 SOUND 7,56:SOUND 8,15
10 FOR I=255 TO 0 STEP -1
20 SOUND 0,I
30 NEXT I
```

```
10   FOR N = 1 TO 100
20   PRINT  PEEK ( - 16336).
30   NEXT N
```

O Apple II padrão não têm comandos intrínsecos para a produção de efeitos sonoros. Desse modo, o efeito conseguido é apenas uma série de cliques. Já o TK-2000 tem o comando **SOUND**. Para rodar o programa acima no TK-2000, substitua as linhas 10 e 20 assim:

```
10 FOR I=80 TO 15 STEP -2
20 SOUND I,30
```

Este é um exemplo do efeito de som que os programadores usam em jogos para dizer "Você falhou" ou "O alie-

nígena aterrisou''. Ao mesmo tempo, quando se faz uma contagem regressiva, a declaração **STEP** existente no laço **FOR... NEXT** deve ser seguida de um número negativo, destinado a diminuir sua freqüência.

PINTANDO O SETE E OUTROS NÚMEROS

Apenas por divertimento — e valiosa experiência em programação — você pode usar laços **FOR... NEXT** para criar variedades de efeitos gráficos. Aqui está uma amostra:

```
7 CLS 0
10 FOR N=0 TO 63
20 LET M=RND(32)-1
30 LET C=RND(9)-1
40 SET (N,M,C)
50 NEXT N
60 GOTO 10
```

```
10 FOR n=0 TO 21
20 LET m=RND*31
30 INK RND*7+1
```

Os comandos ou palavras-chave em BASIC, tal como PRINT, GOTO, STEP, etc., precisam ser digitados sempre em letras maiúsculas?

Sim, quase sempre. É preciso considerar, no entanto, as especificidades das diversas linhas. Alguns computadores, como os compatíveis com o ZX-81 e o Sinclair Spectrum não deixam margem a dúvidas, pois as palavras-chave são entradas por inteiro, em maiúsculas, ao se pressionar a tecla correspondente.

Outros computadores, como os compatíveis com a linha TK-2000 e os Apple II e Apple II Plus, não têm minúsculas (e nem aceitam comandos que não sejam em maiúsculas, caso você tenha essa opção em sua máquina).

Finalmente, os compatíveis com as linhas MSX e TRS-80 aceitam comandos em minúsculas, mas convertem internamente para maiúsculas todos os comandos em BASIC.

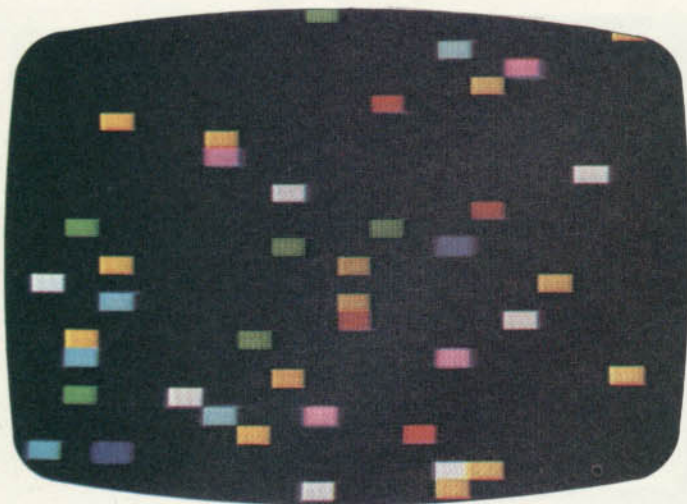
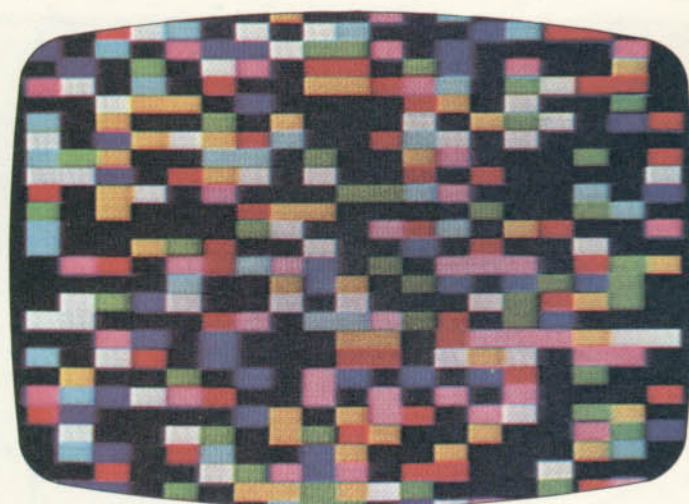
```
40 PRINT AT n,m;"■"
50 NEXT n
60 GOTO 10
```

```
10 SCREEN 3
20 R=RND(-TIME)
30 FOR N=0 TO 191 STEP 4
40 LET M=RND(1)*255
50 LET C=RND(1)*15
60 PSET(M,N),C
70 NEXT N
80 GOTO 30
```

```
5  GR
10  FOR N = 0 TO 39
20  LET M =  RND (1) * 40
30  COLOR=  RND (1) * 15
40  PLOT M,N
50  NEXT N
60  GOTO 10
```

A linha 10 define a altura do desenho que será colocado no vídeo e diz ao computador para imprimi-lo linha por linha.

A linha 20 define a largura e, somada à linha 40, ordena ao computador que imprima pequenos quadrados aleatoriamente naquela largura disponível.

A linha 30 escolhe, ao acaso, as cores dos quadradinhos.

Inicialmente, forma-se na tela uma "colcha de retalhos"...

... que continua a se repetir. Esta é a versão para o TK-90X.

VARIAÇÕES SOBRE UM MESMO TEMA

Experimentar variações desse tema de elaboração de gráficos vai ajudá-lo a se familiarizar com laços **FOR... NEXT** e com a função **RND**.

Aqui temos mais duas para cada máquina. Não se esqueça de digitar o comando **NEW** entre elas, para evitar que os programas se misturem.

O TRS-80 comum e seus compatíveis não selecionam a cor do ponto gráfico; por isso o programa não tem muita graça. Se você quiser ver como é, substitua na versão abaixo a linha 40 e retire a linha 30:

```
40 SET (N,M)
```

```
8 CLS0
10 FOR N=1 TO 60
20 FOR M=0 TO 31
30 LET C=RND(9)-1
40 SET (N,M,C)
45 NEXT M
50 NEXT N
60 GOTO 10
```

Tente também a seguinte variação para o TRS-Color:

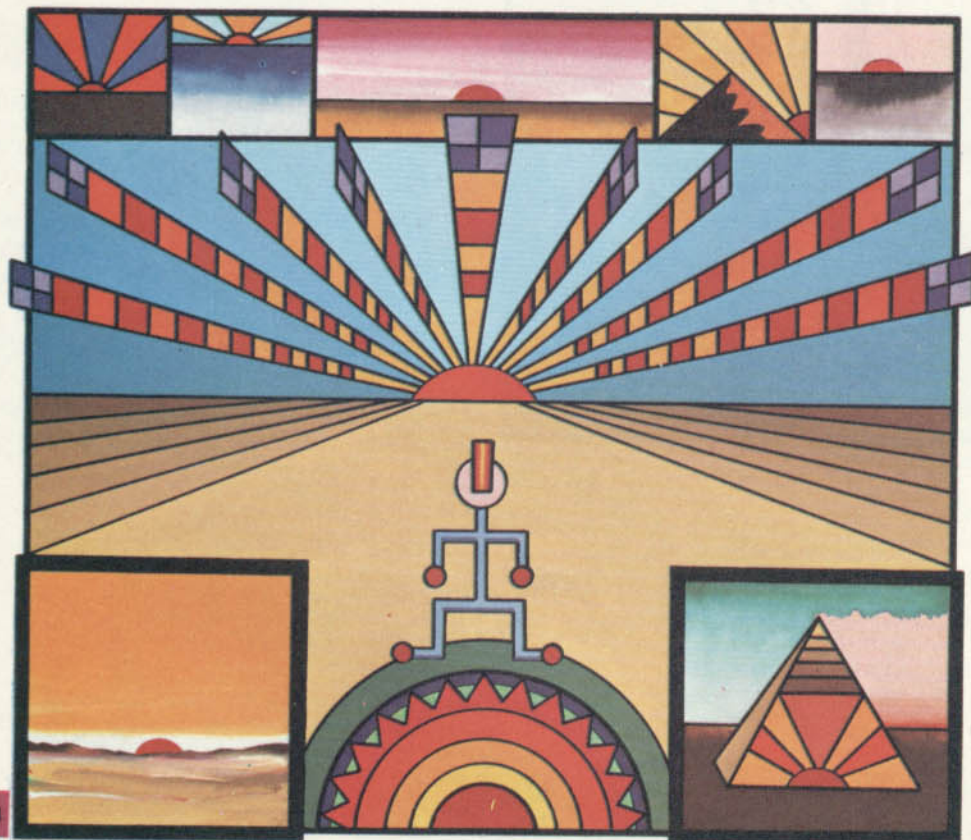
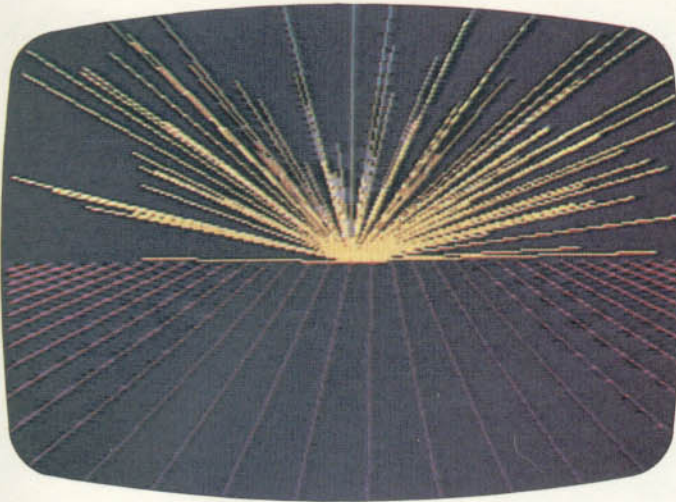
```
8 CLS0
10 LET N=RND(60)
20 FOR M=0 TO 31
30 LET C=RND(9)-1
40 SET (N,M,C)
45 NEXT M
60 GOTO 10
```

```
10 FOR n=0 TO 21
20 FOR m=0 TO 31
30 INK RND*7+1
40 PRINT AT n,m;"■"
45 NEXT m
50 NEXT n
60 GOTO 10
```

Da mesma forma, o ZX-81 não tem cor. Para rodar o programa, retire a linha 30 e use apenas letras minúsculas, quando digitá-lo.

Para o TK-90X, tente ainda a seguinte variação do programa. O que acontece de diferente?

```
10 LET n=RND*21
20 FOR m=0 TO 31
30 INK RND*7+1
40 PRINT AT n,m;"■"
45 NEXT m
69 GOTO 10
```

```
10 SCREEN 3
```

Pôr-do-sol, de autoria do laço *FOR... NEXT* para o TK-2000.

A versão do Apple II para o Caleidoscópio.

```
20 R=RND(-TIME)
30 FOR N=0 TO 191 STEP 4
40 FOR M=0 TO 255 STEP 4
50 LET C=RND(1)*15
60 PSET(M,N),C
65 NEXT M
70 NEXT N
80 GOTO 30
```

```
10  GR
20  LET N =  RND (1) * 40
30  FOR M = 0 TO 39
40  COLOR=  RND (1) * 15
50  PLOT M,N
60  NEXT M
80  GOTO 20
```

Antes de ir adiante, aqui está uma pequena experiência que você deveria tentar — apenas no TK90X e no CP400. Apague a linha 45 dos dois primeiros programas e rode-os novamente com a seguinte modificação:

```
55 NEXT M
```

Você descobriu mais uma característica importante dos laços **FOR... NEXT**: quando se tem dois ou mais laços no mesmo programa, estes devem ficar ou "aninhados" um no outro ou separados. Se eles se cruzam, o programa não funciona.

PÔR-DO-SOL

Este programa usa um laço **FOR... NEXT** para criar um padrão de "pôr-do-sol". A primeira parte do programa define um ponto no meio da tela. Em seguida, ele desenha linhas luminosas que irradiam desse foco.

A segunda parte desenha uma série de linhas na metade inferior do vídeo, partindo de pontos fixos no "horizonte". Isso cria um belo efeito de perspectiva que pode ser aproveitado em programas futuros.

```
20 PMODE 3,1
30 PCLS 3
40 COLOR 2
50 SCREEN 1,0
60 FOR N=1 TO 80
70 LINE(127,95)-(256-RND(256),9
6-RND(96)),PSET
80 NEXT N
90 COLOR 4
100 FOR N=95 TO 191 STEP 12
110 LINE (127,95)-(0,N),PSET
120 LINE (127,95)-(255,N),PSET
130 NEXT N
140 FOR N=0 TO 255 STEP 10
150 LINE(127,95)-(N,191),PSET
160 NEXT N
170 GOTO 170
```

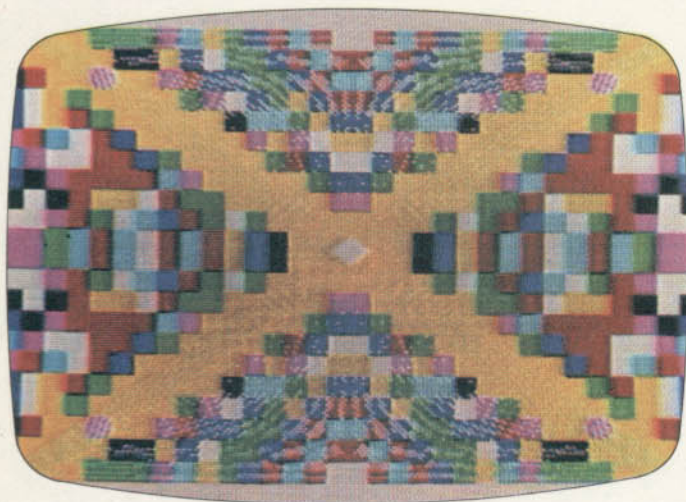
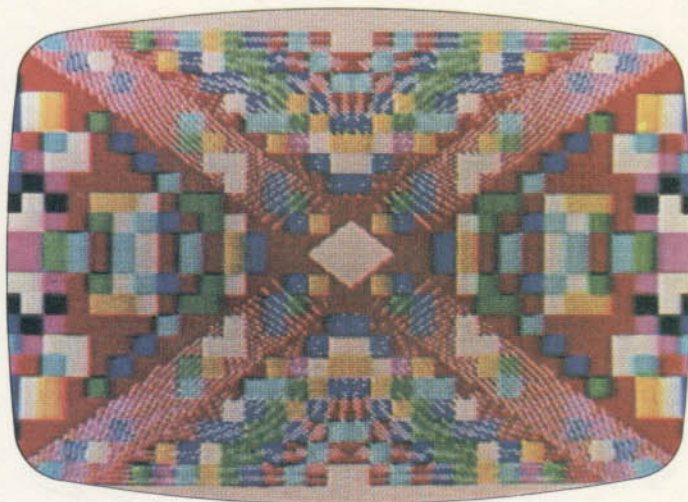
```
  5 BORDER 0: PAPER 0: INK 6:
  CLS
 10 FOR n=1 TO 80
 20 PLOT 127,75
 30 DRAW INT (RND*250)-125,INT
  (RND*97)
 40 NEXT n
 45 INK 5
 50 FOR n=75 TO 0 STEP -15
 60 PLOT 127,75
 70 DRAW -127,-n: PLOT 127,75:
 DRAW 127,-n
 80 NEXT n
100 FOR n=-127 TO 127 STEP 20
110 PLOT 127,75
120 DRAW n,-75
130 NEXT n
```

```
20 SCREEN 2
30 CLS
40 COLOR 11
50 R=RND(-TIME)
60 FOR N=1 TO 80
70 LINE (127,95)-(256-RND(1)*25
6,96-RND(1)*96)
80 NEXT N
90 COLOR 6
100 FOR N=95 TO 191 STEP 12
110 LINE (127,95)-(0,N)
120 LINE (127,95)-(255,N)
130 NEXT N
140 FOR N=0 TO 255 STEP 10
150 LINE (127,95)-(N,191)
160 NEXT N
170 GOTO 170
```

```
10   HGR
20   HCOLOR= 5
30   FOR I = 1 TO 80
40   HPLOT 139,79 TO  RND (1) *
279, RND (1) * 79
50   NEXT I
60   HCOLOR= 6
70   FOR I = 79 TO 159 STEP 10
80   HPLOT 139,79 TO 0,I
90   HPLOT 139,79 TO 279,I
100   NEXT I
110   FOR I = 0 TO 279 STEP 20
120   HPLOT 139,79 TO I,159
130   NEXT I
```

CALEIDOSCÓPIOS

Finalmente, temos um programa espetacular que funciona em quatro de nossas máquinas.

```
3 PMODE 3,1
6 PCLS
```

No TK-80X o Caleidoscópio é mais grosseiro e intenso

... mas seu padrão de formas e cores é sempre cambiante.

```
9 SCREEN 1,0
10 FOR L=0 TO 255 STEP 2
15 COLOR RND(4)
20 LINE(0,0)-(L,191),PSET
30 LINE (255,0)-(255-L,191),PSE
T
40 LINE(0,191)-(L,0),PSET
50 LINE (255,191)-(255-L,0),PSE
T
60 NEXT L
70 GOTO 10
```

```
10 FOR n=0 TO 255 STEP 3
15 INK RND*8
20 PLOT 0,0: DRAW n,175
30 PLOT 255,0: DRAW -n,175
40 PLOT 0,175: DRAW n,-175
50 PLOT 255,175: DRAW -n,-175
```

USE O LAÇO CERTO

Existem muitas oportunidades para usar os comandos **FOR... NEXT** em laços, mas também muitas ocasiões em que você não deve empregá-los.

Quando você quiser, por exemplo, forçar o programa a realizar um número fixo de repetições, sem interrupção em quaisquer fases do processamento, use um laço **FOR... NEXT**.

Em contrapartida, se você quiser que uma seqüência de programa seja executada até que uma determinada condição seja atingida, para em seguida "sair" de dentro do laço, use uma declaração diferente, no lugar de **FOR... NEXT**. O comando mais recomendável em casos assim é o **GOTO**, em BASIC.

```
60 NEXT n
70 GOTO 10
```

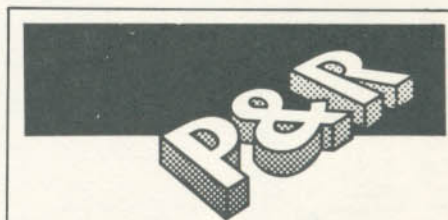
```
3 R=RND(-TIME)
6 CLS
9 SCREEN 2
10 FOR L=0 TO 255 STEP 2
15 COLOR RND(1)*15
20 LINE (0,0)-(L,191)
30 LINE (255,0)-(255-L,191)
40 LINE (0,191)-(L,0)
50 LINE (255,191)-(255-L,0)
60 NEXT L
70 GOTO 10
```

```
10   HGR
20   FOR N = 0 TO 279 STEP 5
30   HCOLOR= RND (1) * 6 + 1
40   HPLOT 0,0 TO N,159
50   HPLOT 279,0 TO 279 - N,159
60   HPLOT 0,159 TO N,0
70   HPLOT 279,159 TO 279 - N,0
80   NEXT N
90   GOTO 20
```

Cada um dos quatro segmentos desses padrões começa com um ponto em um canto do vídeo. O que o laço **FOR... NEXT** faz é contar até o outro lado do vídeo, enquanto um padrão de linhas é desenhado entre os dois pontos criados dessa forma.

A explicação exata de como os gráficos funcionam está em um capítulo posterior, mas apague as linhas 30 e 40 e você terá uma idéia geral do que acontece.

Você pode experimentar, ainda, digitando algumas das linhas responsáveis pelo desenho, variando as cores, mudando os **STEP** na linha 10 e excluindo o **GOTO** na linha final. Em minutos você pode criar centenas de padrões diferentes, um verdadeiro caleidoscópio de formas e cores.

Ao fazer tudo isso, você não estará só criando belas imagens no vídeo. Você estará também se familiarizando com uma das mais úteis "ferramentas" do programador.

Como devo proceder para manter um registro de todas as variáveis usadas em meus programas, sem me perder entre tantos Xs e Ys?

Uma forma de evitar o uso desordenado de variáveis em um programa é dar-lhes nomes que lembrem o que elas representam (nomes mnemônicos). Por exemplo, se uma variável é usada para armazenar o número de espaçonaves abatidas em um jogo, podemos "batizá-la" de **NAVES**.

É fácil manter um registro de variáveis num programa curto. No caso de programas mais longos, convém explicitar mais claramente certas funções. Assim, utilize **FOR LINHA = ...** e **FOR COLUNA = ...** (se você estiver colocando coisas na tela) em vez de **FOR X = ...** e **FOR Y = ...**

Alguns computadores, como o TRS-80 e o Apple II, reconhecem apenas as duas primeiras letras de uma variável, descartando as demais. Neste caso, tente utilizar iniciais que funcionem como abreviaturas adequadas, como **T** para "tempo", **RC** para "recorde", **L** para "linha", e assim por diante.

COMO O COMPUTADOR É CAPAZ DE REPETIR TRECHOS DE UM PROGRAMA

A capacidade de repetir de forma controlada o mesmo trecho de um programa pode ser encontrada em todas as linguagens de programação. No BASIC, a programação de laços é muito facilitada pelos comandos **FOR**, **STEP** *e* **NEXT**, *explicados nesta lição; porém eles não são essenciais.*

É possível programar laços de repetição de outro modo, simplesmente usando-se uma variável contadora e uma linha com a declaração (ou comando) **IF**.

Dessa forma, a programação direta de repetições exemplifica de modo mais claro e objetivo a maneira que o computador usa, internamente, para repetir trechos de um programa.

Suponhamos que o programador queira imprimir dez vezes a mesma mensagem na tela, usando o comando **PRINT**. *Inicialmente, é preciso definir e igualar a 0 uma variável contadora (por exemplo,* **N**). *Em seguida, a cada repetição do laço, essa variável é incrementada, somando-se o número 1 ao valor anterior (este número corresponde, no caso, ao valor* **STEP** *no comando* **FOR**). *Finalmente, uma linha adicional do programa deve testar se esse valor de* **N** *é maior do que 10. Se não for, o laço será repetido por meio de um comando* **GOTO** *e enviará o programa para trás. Se for maior do que 10, o programa terminará ou prosseguirá para outro trecho.*

APONTAR ...FOGO!

Jogos de ação dependem da capacidade do jogador em controlar os movimentos na tela. Veja como fazer isso e aprenda a disparar mísseis, integrando tudo num jogo divertido.

Os videogames para microcomputadores, como o *Space Invaders*, não teriam nenhuma graça se a movimentação e os disparos do canhão laser não pudessem ser controlados pelo jogador simultaneamente com o desenrolar do jogo. Esse tipo de controle através do teclado ou do *joystick* é uma característica de todos os jogos de ação, mesmo os mais simples.

O elemento mais importante dessa técnica de programação é como fazer para que o computador perceba automaticamente quando uma tecla qualquer é pressionada, sem que seja necessário acionar todas as vezes as teclas <**ENTER**> ou <**RETURN**>. Feito isso, o programa pode tomar decisões sobre que gráfico animar, que "disco voador" destruir, etc. Outro aspecto essencial é que, enquanto o jogador não acionar os controles de jogo, o computador poderá se dedicar à movimentação contínua de gráficos e figuras sobre a tela, usando as técnicas aprendidas na última lição de programação de jogos.

COMO DETECTAR TECLAS PRESSIONADAS

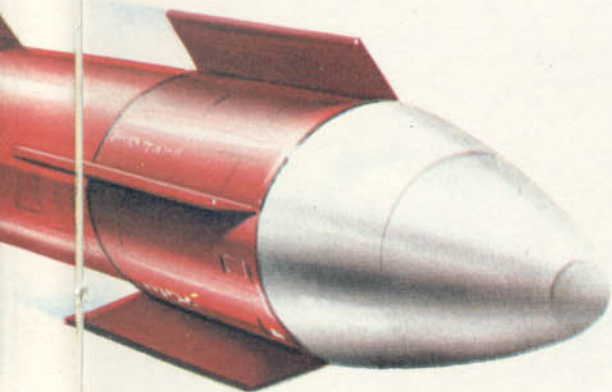
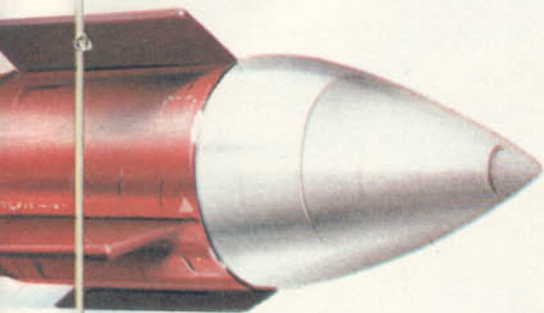
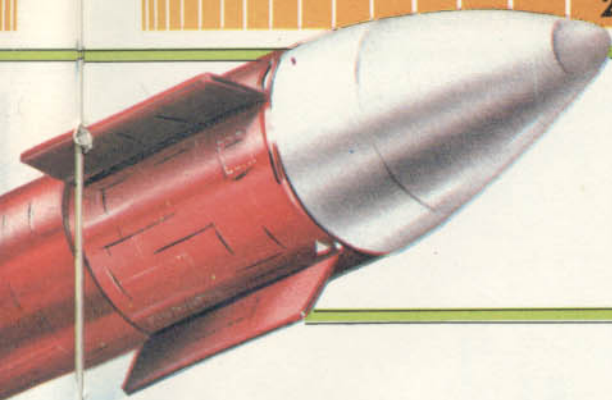
Em princípio, todos os micros usam o mesmo método para detectar se uma tecla foi pressionada. Mas, como os comandos do BASIC para essa função não foram padronizados, há uma grande variabilidade entre os diversos tipos de computadores.

TTSSM

Para a maioria dos computadores que utilizam o interpretador da linguagem Microsoft BASIC (adotada nos micros da linha TRS-80, Sinclair e MSX), a função utilizada para detectar se uma tecla foi pressionada chama-se **INKEY$**. Ao encontrar essa função em um programa, o computador executa uma "varredura" do teclado, para ver se alguma passou do estado de desligada para ligada, e retorna o caractere ao programa através da função **INKEY$**. Se nada foi pressionado, a função volta com um valor "vazio".

Por exemplo:

- APRENDA A DETECTAR SE UMA TECLA FOI PRESSIONADA
- DISPARE SEU MÍSSIL
- COMO CONTROLAR UMA FIGURA EM MOVIMENTO
- CONSTRUA BLOCOS GRÁFICOS
- DESTRUA O INIMIGO
- BASES DE MÍSSEIS
- A AUTO-REPETIÇÃO
- COMO MONTAR ALVOS MÓVEIS

TT

```
20 CLS
30 LET A$=INKEY$:IF A$="" THEN
GOTO 30
40 PRINT @269,"OPA!"
```

SS

```
20 CLS
30 IF INKEY$="" THEN  GOTO 30
40 PRINT AT 11,14;"Ai!"
```

```
20 CLS
30 LET A$=INKEY$:IF A$="" THEN
GOTO 30
40 LOCATE 18,11:PRINT "Ui !"
```

Execute o programa e pressione qualquer tecla, à exceção de **<CAPS SHIFT>** ou **<SYMBOL SHIFT 1>** (nos Sinclair), **<BREAK>** ou **<SHIFT>** (nos TRS-80) ou **<CTRL STOP>** ou **<SHIFT>** (no MSX). O computador vai exibir um "grito de dor" no meio da tela. O programa funciona da seguinte maneira:

A linha 20 limpa a tela. A linha 30 faz com que o computador espere até que uma tecla seja acionada para continuar o programa. Note que não há espaços entre as aspas. Por causa disto, a linha 30 quer dizer: "Se **INKEY$** = nada, ou se nenhuma tecla está sendo pressionada, cheque novamente". É muito importante ter o **IF...THEN GOTO 30** porque de outra forma o computador verificaria apenas uma vez o teclado e só por uma fração de segundo.

Assim que uma tecla é acionada, **INKEY$**é igualado à tecla. Por exemplo, se "3" é pressionado, então **INKEY$** = "3". Isso basta para que a linha 40 imprima "Ai!" na tela.

Na maioria dos jogos você deve pressionar uma tecla para mover um tanque, uma míssil, etc. Se você mudar a linha 40, verá como isto é feito (no TK-85 use um "D" maiúsculo):

TT

```
40 IF A$="D" THEN PRINT @264,"H
UM, ISSO E BOM!" ELSE PRINT @26
9,"OPA!"
```

S

```
40 IF INKEY$="d" THEN  PRINT
AT 11,9;"Hum, isso e bom!":
STOP
50 PRINT AT 11,14;"Ai!"
```

MSX

```
20 CLS
30 LET A$=INKEY$:IF A$="" THEN
GOTO 30
40 IF A$="D" OR A$="d" THEN LOC
ATE 10,11:PRINT "mmm... isso é
bom !":END
50 LOCATE 18,11:PRINT "Ui !"
```

A linha 40 agora verifica se a tecla "D" foi pressionada, ou seja, se **INKEY$**é igual a "D" ou a "d". Se não for, o programa para os micros da linha Sinclair, que não têm a declaração **ELSE**, vão ignorar a linha 40 e passar para a linha 50. Qualquer outra tecla diferente de "D" mostrará, quando pressionada, a segunda mensagem de "reação emocional". Nos micros de outras linhas, o **IF...THEN...ELSE** permite programar em uma única linha as duas alternativas.

Existem mais duas exigências importantes nesse programa. A primeira é que o "D" ou "d" precisa estar entre aspas, ou o computador o interpretará como uma variável. A segunda vale apenas para o TK-90X: nessa linha de micros, deve-se usar um "d" minúsculo; caso contrário você deve pressionar **<CAPS SHIFT>** e "d" para que o programa funcione.

Tanto nesse programa quanto no anterior é possível notar que todos os computadores utilizam exatamente a mesma técnica de investigação do teclado. Apenas a maneira de colocar uma mensagem no meio da tela é diferente para cada um.

DISPARE SEU MÍSSIL

Agora você verá que a partir da detecção do pressionamento de uma tecla usando o **IF INKEY$** = "tecla" é muito fácil disparar um míssil ou mover uma base de foguetes. Este programa dispara um míssil de uma base quando a tecla "F" é acionada:

T

```
20 CLS
30 PRINT @397,"###"
40 LET K$=INKEY$:IF K$=""THEN G
OTO 40
50 IF K$="F" THEN LET M=334 ELS
E GOTO 40
60 PRINT @M,"↑"
70 PRINT @M+32," "
80 LET M=M-32
90 IF M>0 THEN GOTO 60
100 GOTO 20
```

```
10 CLS
20 PRINT @797,"###"
30 K$=INKEY$ : IF K$="" GOTO 30
```

```
40 IF  K="F"   THEN M=374  ELSE
   GOTO 30
50 PRINT @M,CHR$(94);
60 PRINT @M+64," ";
70 M=M-64
80 IF M>0 GOTO 50
90 GOTO 10
```

Para rodar também no ZX-81, digite o programa abaixo apenas com letras maiúsculas:

```
20 CLS
30 PRINT AT 21,14;"   "
40 IF INKEY$="" THEN  GOTO 40
50 IF INKEY$<>"f" THEN  GOTO 40
55 LET y=20
60 PRINT AT y,15;"↑"
70 LET y=y-1
75 PAUSE 1
80 PRINT AT y+1,15;" "
90 IF y>0 THEN  GOTO 60
```

MSX

```
20 CLS
30 LOCATE 17,20:PRINT "   "
40 LET K$=INKEY$:IF K$="" THEN
GOTO 40
50 IF K$="F" OR K$="f" THEN LET
 M=20 ELSE GOTO 40
60 LOCATE 18,M:PRINT" "
70 LOCATE 18,M+1:PRINT " "
80 LET M=M-1
90 IF M>0 THEN GOTO 60
100 GOTO 20
```

A linha 30 imprime uma base de mísseis formada de três símbolos (no Sinclair e no MSX, três símbolos gráficos), na parte mais baixa da tela (última linha).

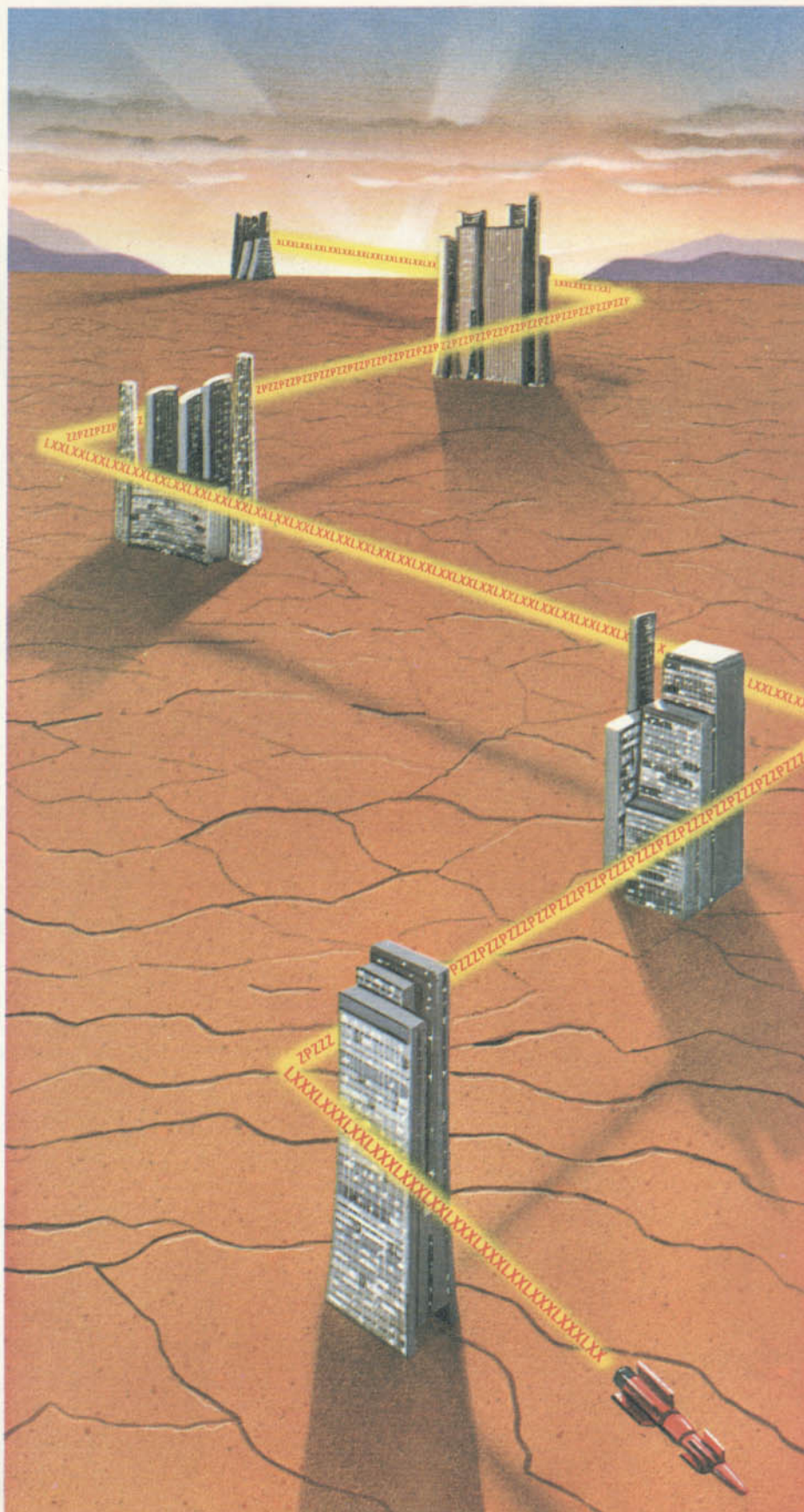
Na linha 40, **LET K$ = INKEY$** é muito importante, visto que você quer checar se a tecla foi pressionada várias vezes durante o programa. Esta não funcionaria sem essa declaração. O computador se recorda do valor de **INKEY$** apenas por uma fração de segundo e, se você não checar **INKEY$** suficientemente rápido, ele pode esquecer que a tecla foi pressionada.

Mas, você pode também fazer com que ele se lembre do que foi teclado dando um nome, **K$**, à tecla que foi pressionada e checando mais tarde (versão para a linha TRS), ou dando um novo comando **INKEY$**.

A linha 50 verifica se "F" foi pressionado. Se não, o programa volta à linha 40 e continua a varrer o teclado. "M" é a posição do míssil depois de ter sido disparado.

A linha 60 imprime o míssil na tela e a linha 70 se encarrega de apagá-lo de suas posições anteriores.

A linha 80 faz o míssil se deslocar pa-

ra cima na tela, e a linha 90 evita que o computador tente imprimir o míssil numa posição fora do vídeo, o que provocaria uma mensagem de erro. A primeira posição de tela tem o número 0, e o computador não pode imprimir nada em uma posição de número menor que 0. Quando M = 0 o programa é reiniciado. A técnica a ser utilizada para fazer o míssil subir vai depender, agora, do formato da tela do computador utilizado.

A linha 80 subtrai um número constante N da posição anterior do míssil (variável **M**), de maneira que ele avança uma linha em direção ao topo da tela a cada vez que é impresso. Esse número vale 32 para os micros compatíveis com o TRS-Color, pois há 32 colunas na tela do computador. Já para o TRS-80, o número é 64.

A linha 80 subtrai 1 da posição anterior do míssil, pois o comando de posicionamento utilizado, o **PRINT AT**, identifica as coordenadas (linha, coluna) onde serão exibidos os gráficos. Assim, a coluna é mantida constante, para o míssil subir "reto", na coluna 15, e o que varia é Y, ou a altura dele (lembre-se de que Y = 0 está no topo da tela, e não embaixo.

O comando de posicionamento é o **LOCATE**, que dá as coordenadas para posicionamento. A linha 80 tem por função diminuir em 1 o valor anterior do míssil (variável **M**), para que ele se desloque para cima, na mesma coluna (constante 18).

MOVIMENTOS SOBRE A TELA

Da forma que está, o programa da base de mísseis é bastante limitado para um jogo de "artilharia", mas, imprimindo movimento lateral à base conseguiremos melhorá-lo um pouco. Vejamos como fazer isso:

```
20 CLS
30 LET P=397
40 PRINT @P,CHR$(143)+CHR$(140)
+CHR$(128)+CHR$(140)+CHR$(143)
50 LET K$=INKEY$:IF K$="" THEN
GOTO 50
60 IF K$="E" THEN LET P=P-1:GOT
O 90
70 IF K$="D" THEN LET P=P+1:GOT
O 90
80 GOTO 50
90 IF P<384 OR P>411 THEN GOTO
50
100 GOTO 40
```

A linha 30 determina a posição inicial da base, e a linha 40 coloca o míssil naquela posição.

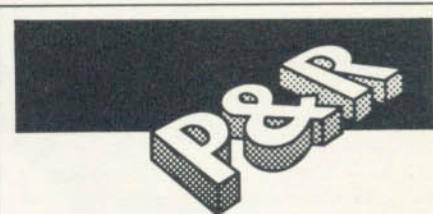
A linha 50 verifica o teclado, fazendo o computador esperar até que uma tecla seja pressionada.

A linha 60 investiga se "L" foi pressionado. Se foi, ela move a base uma posição para a esquerda, subtraindo 1 do número que determina a posição da base. A linha 70 confere se "R" foi pressionado e nesse caso muda a posição da base adicionando 1 à coordenada. A linha 90, finalmente, verifica se o programa está dizendo ao computador para colocar a base fora da tela — em caso afirmativo, o programa retorna à linha 50.

```
20 CLS
30 P=797
40 PRINT@P,CHR$(32)+CHR$(184)+
   CHR$(191)+CHR$(180)+CHR$(32)
50 K$=INKEY$ : IF K$="" GOTO 50
60 IF K$="L" THEN P=P-1
70 IF K$="R" THEN P=P+1
80 GOTO 50
90 IF P<767 OR P>828 GOTO 50
100 GOTO 40
```

O programa para o TRS-80 funciona de modo idêntico ao programa para o TRS-Color. Note que os caracteres gráficos têm símbolos diferentes, bem como as coordenadas de posicionamento dos comandos **PRINT @**.

Para rodar também no ZX-81, digite o programa abaixo apenas com letras maiúsculas. Além disso, ponha os comandos **LET** da linha 80 em duas linhas separadas.

```
 30 CLS
 40 LET x=15
 50 LET y=13
 60 PRINT AT y,x;"▟▙"
 70 IF INKEY$="" THEN  GOTO 70
 80 LET lx=x: LET ly=y
 90 PRINT AT ly,lx;"   "
100 IF INKEY$="q" THEN  STOP
110 IF INKEY$="w" THEN  LET y=
y-1
120 IF INKEY$="z" THEN  LET y=
y+1
130 IF INKEY$="a" THEN  LET x=
x-1
140 IF INKEY$="s" THEN  LET x=
x+1
150 IF x<1 OR x>29 THEN  LET x
=lx
160 IF y<1 OR y>20 THEN  LET y
=ly
170 GOTO 60
```

Posso escolher qualquer tecla para operar os controles de um jogo no microcomputador?

Sim. Tudo o que você tem a fazer é mudar os caracteres nas linhas do programa que testam qual foi a tecla pressionada e detectada por **INKEY$**. Mas cuidado: o que parece ser bastante lógico na teoria às vezes funciona mal na prática. Por exemplo, usar as teclas E, D, C, B para representar esquerda, direita, em cima e embaixo, dificulta as ações do jogador. Por isso, costumam-se usar as teclas de movimentação do cursor, principalmente nos computadores onde elas estão aglomeradas no mesmo ponto do teclado, ou presentes em pares complementares. Outra dica é usar A e Z, para movimentar para cima e para baixo, e K e L para direita e para esquerda, pois ocupam posições opostas no teclado. Já a barra de espaços é muito boa para efetuar disparos, pois o polegar pode se apoiar nela.

As linhas 40 e 50 estabelecem a posição inicial da base de mísseis, na 13ª linha da tela, e quinze espaços à esquerda. A linha 60 mostra a base na tela.

A linha 70 faz o computador esperar até que alguma tecla seja impressa. Se foi, a base é movida.

As linhas 80 e 90 "perseguem" a base de mísseis em movimento, colocando três espaços em branco sobre sua última localização; de modo a apagá-la. Como a linha 60 está dentro do laço de repetição e sempre vem antes da linha 90, a base é sempre impressa em sua nova posição antes que a posição antiga seja apagada.

A linha 100 termina o programa se a letra Q (de *quit*, em inglês, que significa abandone) for pressionada.

A linha 110, por sua vez, verifica se a tecla "W" foi acionada, subtraindo 1 do valor de "Y" em caso afirmativo. O efeito disso é a movimentação da base uma linha para cima.

As linhas 120 a 140 operam de forma similar. A linha 120 move a base para baixo se "Z" é pressionado.

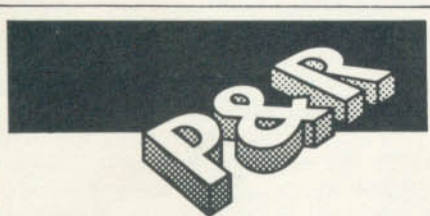
A linha 130, move para a esquerda,

caso "A" seja acionado, e a 140 para a direita, ao se pressionar "S". A linha 150 evita que a base seja colocada fora das laterais da tela.

A linha 160 impede que a base saia por cima ou por baixo da tela. Neste caso, fazer **Y = LY** resolve o problema. A linha 170 fecha o laço, reiniciando todo o processo.

```
30 CLS
40 LET X=17
50 LET Y=20
60 LOCATE X,Y:PRINT " ▄█▄ "
70 A$=INKEY$:IF A$="" THEN GOTO
 70
80 LET LX=X:LET LY=Y
90 LOCATE LX,LY:PRINT"     "
100 IF A$=CHR$(27) THEN END
110 IF A$=CHR$(28) THEN X=X+1
120 IF A$=CHR$(29) THEN X=X-1
130 IF A$=CHR$(30) THEN Y=Y-1
140 IF A$=CHR$(31) THEN Y=Y+1
150 IF X<0 OR X>36 THEN X=LX
160 IF Y<0 OR Y>20 THEN Y=LY
170 GOTO 60
```

O programa para a linha MSX é bastante compacto e elegante. A linha 60 coloca a base em posição inicial (três caracteres gráficos), que é definida nas linhas 40 e 50, ou seja, na linha 20 e na coluna 17 (no meio da última linha da tela). A linha 70 detecta se alguma tecla foi pressionada.

Se alguma tecla foi pressionada, o programa armazena em **LX** e **LY** as coordenadas da última posição da base (X e Y), para usá-las posteriormente (linha 80), e apaga a base nessa posição, por meio da linha 90.

Por que o meu programa "bomba" (isto é, falha) se, num jogo qualquer, a figura que está sendo movimentada atinge a borda da tela?

Provavelmente porque um dos valores que controlam o posicionamento da impressão do gráfico na tela tornou-se muito grande ou muito pequeno, excedendo os limites da tela. Portanto, verifique de novo as linhas que fazem essa tarefa, caso você obtenha alguma mensagem de erro do computador, neste sentido. Por precaução, é sempre recomendável testar os valores de posicionamento antes de imprimir alguma coisa na tela: se eles excederem os limites, podem ser corrigidos para ficarem dentro da tela.

As linhas 100 a 140 detectam se a tecla apertada foi uma das seguintes: flecha para cima, flecha para baixo, flecha para esquerda e flexa para direita (teclas de controle do cursor), que são identificadas pelos seus códigos respectivos (29, 28, 30 e 31). O jogo termina quando a tecla **CLEAR** (código 26) tiver sido pressionada.

As linhas 150 e 160 verificam se as bordas da tela foram ultrapassadas. Se foram, retornam à posição da base para as últimas coordenadas (**LX** e **LY**).

Finalmente, a linha 60 faz o ciclo se repetir, para a próxima pressão à tecla.

FAÇA O SEU JOGO

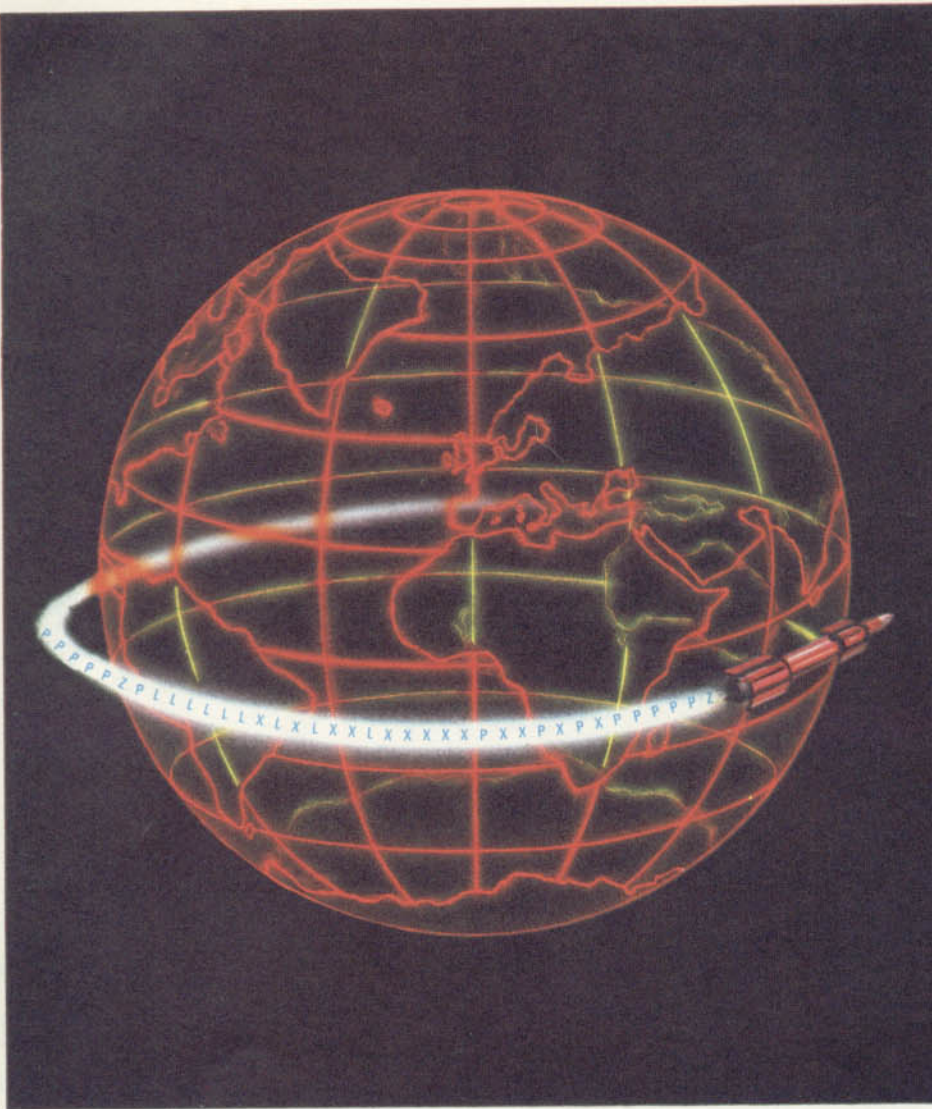
Agora você já tem alguns blocos a partir dos quais podem ser montados vários jogos. O exemplo abaixo mostra como usá-los para desenvolver um jogo bem simples, em que um canhão móvel dispara mísseis em direção a um alvo (uma estrela):

```
20 CLS
30 FOR N=1 TO 100:NEXT N
40 LET PO=430
50 LET B$=CHR$(143)+CHR$(140)+C
HR$(128)+CHR$(140)+CHR$(143)
60 LET A=RND(30)+64
70 PRINT @A,"*"
80 LET LP=PO
90 PRINT @PO,B$
100 IF PEEK(340)=247 THEN LET P
O=PO-1
110 IF PEEK(338)=247 THEN LET P
O=PO+1
120 IF PO<415 OR PO>444 THEN LE
T PO=LP
130 LET K$=INKEY$
140 IF K$="F" THEN LET M=PO-30
ELSE GOTO 80
150 PRINT @M,"↑";
160 PRINT @M+32," ";
170 LET M=M-32
180 IF M=A THEN GOTO 20
190 IF M>0 THEN GOTO 150 ELSE P
RINT @M+32," ";
200 GOTO 80
```

```
20 CLS
30 FOR N=1 TO 100:NEXT N
40 PO=800
50 B$=CHR$(32)+CHR$(184)      +
   CHR$(191)+CHR$(180)        +
   CHR$(32)
60 A=RND(60)+2
70 PRINT@A,"*";
80 LP=PO
90 PRINT@PO,B$;
100 IF PEEK(14344)<>0 THEN  PO=
    PO-1
110 IF PEEK(14368)<>0 THEN  PO=
    PO+1
120 IF  PO<767  OR PO>828  THEN
    PO=LP
130 K$=INKEY$
140 IF K$="F" THEN M=PO-62 ELSE
    GOTO 80
150 PRINT@M,CHR$(94);
160 PRINT@M+64," ";
170 M=M-64
180 IF M=A GOTO 20
190 IF  M>0 THEN GOTO 150  ELSE
    PRINT @M+64," ";
200 GOTO 80
```

Para rodar também no ZX-81, digite o programa abaixo apenas com letras maiúsculas:

```
 20 CLS
 30 PAUSE 25
 40 LET x=15: LET y=21
 50 LET B$=" ▄█▄ "
 60 LET a=INT (RND*28)+2
 70 PRINT AT 2,a;"*"
 80 LET xx=x
 90 PRINT AT y,x;B$
100 IF INKEY$="z" THEN  LET x=
x-1
110 IF INKEY$="x" THEN  LET x=
x+1
120 IF x<0 OR x>27 THEN  LET x
=xx
140 IF INKEY$<>"f" THEN  GOTO
80
145 LET m=y-1
150 PRINT AT m,x+2;"↑"
160 PRINT AT m+1,x+2;" "
170 LET m=m-1
180 IF m=2 AND x+2=a THEN
GOTO 20
190 IF m<>1 THEN  GOTO 150
195 PRINT AT m+1,x+2;" "
200 GOTO 80
```

```
20 CLS
40 LET X=17:LET Y=20
50 LET B$=" ▄█▄ "
60 LET A=INT(RND(1)*36)+2
70 LOCATE A,0:PRINT "*"
80 LET XX=X
90 LOCATE X,Y:PRINT B$
95 LET A$=INKEY$:IF A$="" THEN
GOTO 95
100 IF A$=CHR$(28) THEN LET X=X
+1
110 IF A$=CHR$(29) THEN LET X=X
-1
120 IF X<0 OR X>35 THEN LET X=X
X
130 IF A$<> " " THEN GOTO 80
140 LET M=Y-1
150 LOCATE X+2,M:PRINT "█"
160 LOCATE X+2,M+1:PRINT " "
170 LET M=M-1
180 IF M=0 AND X+2=A THEN GOTO
20
```

```
190 IF M<>0 THEN GOTO 150
200 LOCATE X+2,M+1:PRINT" "
210 GOTO 80
```

Quando você executar o programa, verá uma estrela perto do topo da tela. As teclas Z e X movem a base de foguetes para a esquerda e a direita, e a tecla F dispara o míssil para destruir a estrela em sua posição.

Pense no programa como tendo três partes: uma, até a linha 70; outra, da 80 até a 120; e a última, da 130/140 até a 200.

As linhas que vão de 130/140 até 200 são similares ao programa anterior que dispara o míssil.

As variáveis e os **GOTO** foram mudados, mas a única linha adicionada foi a 180. Ela verifica se o míssil e a estrela estão no mesmo lugar. O programa recomeça em caso positivo.

A porção central, da linha 80 à 120, é uma versão compacta do programa "Movimentos Sobre a Tela". Os **PEEK** no programa para o CP400 verificam se Z ou X foram acionados e alteram **PO** de maneira apropriada.

A primeira parte do programa tem várias funções. A linha 30 provoca uma pequena pausa antes que o programa continue. Isso é importante quando a linha 180 fecha o laço no fim do programa. As linhas 40 e 50 determinam a posição inicial da base e definem sua forma.

MELHOR MOVIMENTAÇÃO

Pressionar a tecla que move a base para a direita ou para a esquerda toda vez que se precisa deslocar figuras é bastante cansativo. Então, usualmente montamos um mecanismo de auto-repetição.

Alguns micros, como os da linha MSX, possuem auto-repetição em todas as teclas; por isso o programa dado acima não necessita qualquer modificação: basta ficar pressionando a tecla de comando para baixo, que a base se move automaticamente.

Já para os computadores que não dispõem desses recursos, temos que nos satisfazer com o **INKEY$**, mesmo. Mas é difícil programar movimentos contínuos usando **INKEY$**.

Um modo de se resolver esse problema é ilustrado a seguir.

```
20 CLS
30 LET BL$=CHR$(128)
40 LET PO=238
50 PRINT @PO,BL$
60 LET LP=PO
70 IF PEEK(340)=247 THEN LET PO
=PO-1:GOTO 120
80 IF PEEK(338)=247 THEN LET PO
=PO+1:GOTO 140
90 IF PEEK(338)=251 THEN LET PO
=PO-32:GOTO 150
100 IF PEEK(342)=253 THEN LET P
O=PO+32:GOTO 150
110 GOTO 70
120 IF(LP AND 31)=0 THEN LET PO
=LP
130 GOTO 150
140 IF(PO AND 31)=0 THEN LET PO
=LP
150 IF PO>510 OR PO<0 THEN LET
PO=LP:GOTO 70
160 PRINT @LP," ";
170 PRINT @PO,BL$;
180 GOTO 60
```

```
20 CLS
30 BL$=CHR$(191)
40 PO=540
50 PRINT@PO,BL$;
60 LP=PO
70 IF  PEEK(14344)<>0 THEN  PO=
   PO+1:GOTO 120
80 IF  PEEK(14368)<>0 THEN  PO=
   PO-1:GOTO 140
90 IF  PEEK(14352)<>0 THEN  PO=
   PO-64:GOTO 150
100 IF PEEK(14400)<>0 THEN  PO=
    PO+64:GOTO 150
110 GOTO 70
120 IF (LP AND 63)=0 THEN PO=LP
130 GOTO 150
140 IF (PO AND 63)=0 THEN PO=LP
150 IF PO>1022 PR PO<0 THEN PO=
    LP:GOTO 70
160 PRINT@LP," ";
170 PRINT@PO,BL$;
180 GOTO 60
```

Ao executar este programa você terá um bloco posicionado no centro da tela. O programa permite que você o mova para cima e para baixo, para a direita e para a esquerda.

APRENDA A CONTAR COM UM DEDO SÓ

Os computadores contam em binário, como se tivessem milhões de mãos com um dedo só. Para aprender a programar em código de máquina, você deve conhecer esse sistema.

Um dos problemas de se aprender código de máquina é que você precisa entender um pouco da teoria dos números. Não é uma tarefa tão difícil quanto pode parecer. Se você sabe contar até 16, não terá qualquer dificuldade. Mas, inicialmente, é preciso aprender a contar até 2.

POR QUE A BASE 10?

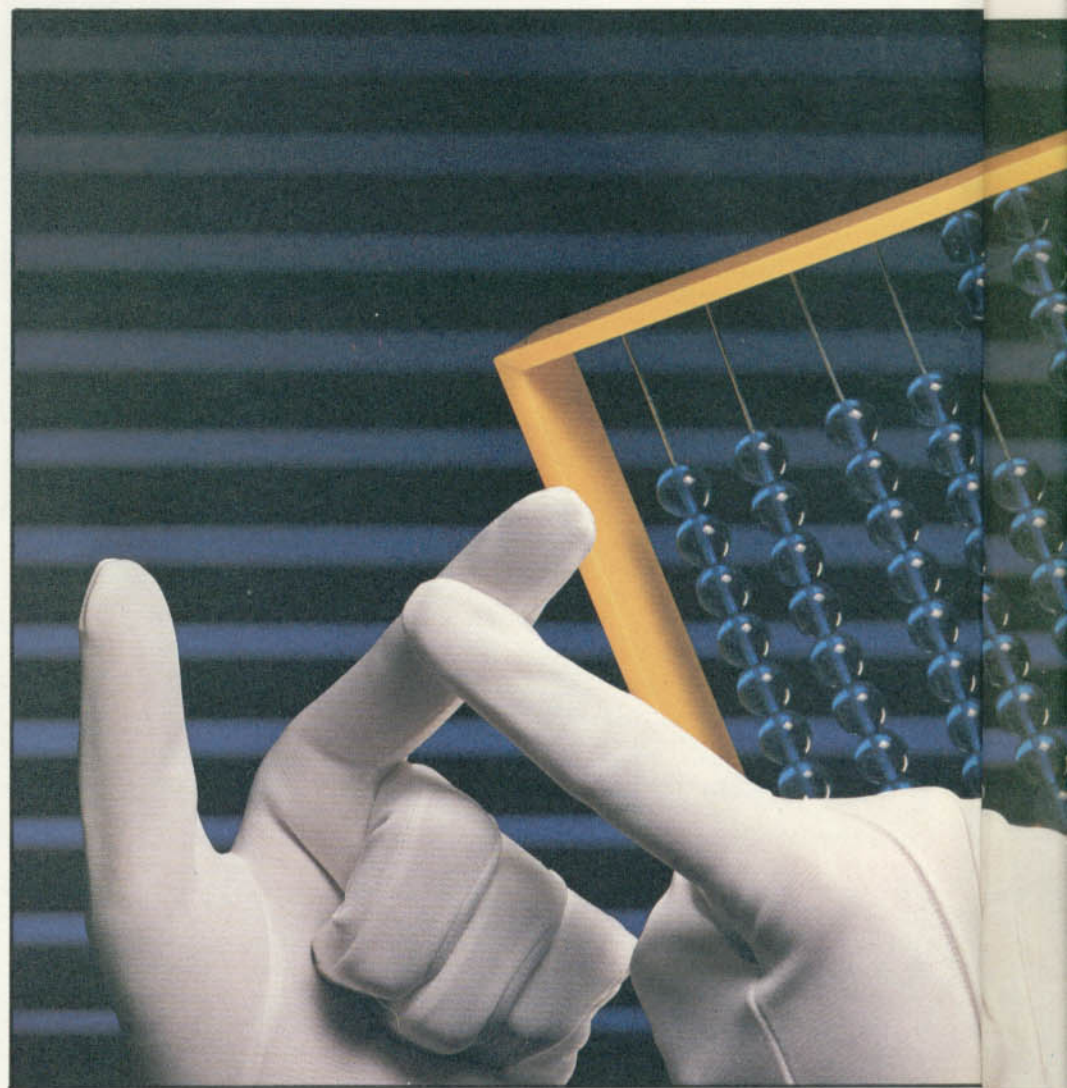
Mesmo a pessoa mais avessa a matemática não tem qualquer dificuldade para dizer as horas ou acompanhar o resultado de uma partida de futebol. O uso dos números faz parte do dia-a-dia de tal forma que nunca nos preocupamos com a maneira com que eles funcionam.

No mundo ocidental usamos habitualmente um sistema numérico baseado no número 10. Isto significa que começamos uma contagem usando os números de 0 a 9. Se precisarmos de uma grandeza maior do que 9 (por exemplo, 10), usamos os dígitos disponíveis. Assim, colocamos o 1 na "casa" da esquerda e o 0 à sua direita.

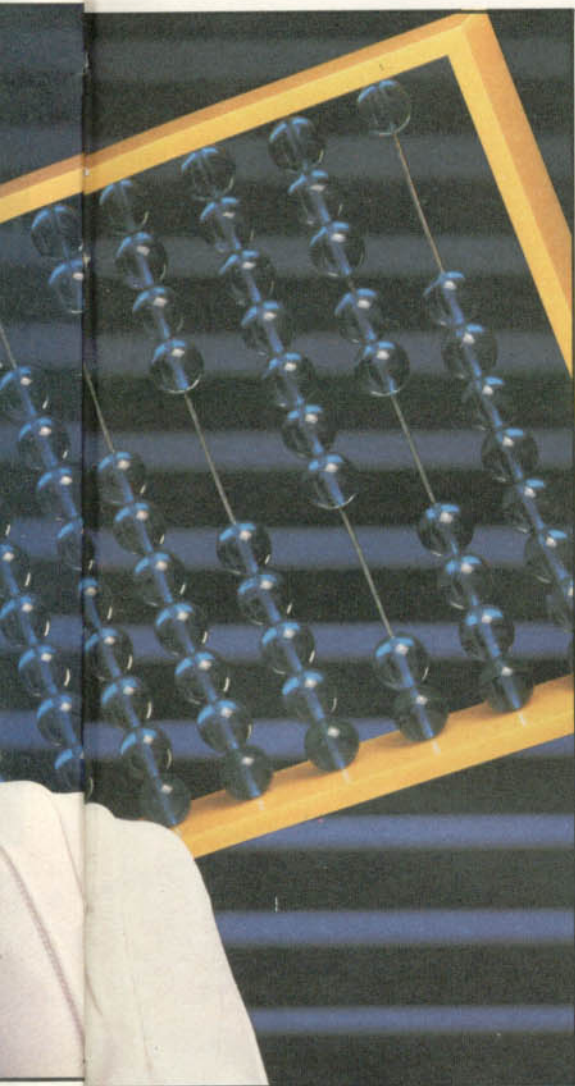
Este é chamado um sistema de numeração de base 10 ou decimal, porque o valor do dígito é multiplicado por 10 a cada posição que contamos, a partir da direita. Por exemplo, o número 3275 vale $5 + 7 \times 10 + 2 \times 10 \times 10 + 3 \times 10 \times 10 \times 10$ ou seja $5 + 10 + 200 + 3000$. Cada dígito multiplica seu valor absoluto por 10 a cada vez que é movido uma posição mais à esquerda. Todo sistema que usa esse método para representar números é chamado de *posicional*.

Tudo isso parece óbvio porque fazemos esses cálculos todos os dias, sem pensar muito neles. Entretanto, existem outros sistemas de numeração, diferentes do decimal, como por exemplo, os que são usados nos modernos micros.

SISTEMAS ANTIGOS E NOVOS

Os antigos babilônios tinham um sistema de numeração baseado no número 60. Os vestígios disso podem ser vistos nas nossas medidas de tempo e de ângulos: existem 60 segundos em cada minuto, 60 minutos em cada hora, assim como 60 minutos de arco em um grau e 6 vezes 60 graus — ou seja, 360 — num círculo completo.

Você pode contar até 59 segundos, mas, se adicionar mais um segundo a esse número, terá um minuto e recomeçará a contar os segundos do 0. E quando tiver 59 minutos e 59 segundos, um segundo a mais fará uma hora, sendo zerados os minutos e segundos.

Resquícios de numeração em outras bases podem ser encontrados em velhos sistemas de medidas da Inglaterra. Existem 8 pintas em cada galão, 12 polegadas em cada pé e 16 onças em cada libra. Outra base que ocorre com freqüência é o número 12. Temos 12 polegadas em um pé, 12 unidades em uma dúzia e 12 dúzias em uma grosa. A base 12 era muito usada tanto para dinheiro como para bens, devido à facilidade com que se divide por 2, 3, 4 ou 6. Certas coisas tinham que ser divididas entre várias pessoas, e a base 12 tornava as transações muito mais simples. O número 10, ao contrário, só é divisível por 2 e 5. Vemos, portanto, que muitas vezes é conveniente usar outra base que

- O SISTEMA DECIMAL
- OUTROS SISTEMAS NUMÉRICOS
- COMO CONTAR EM BASE 9
- CONTAS EM DIFERENTES BASES
- CONVERSOR PARA TODAS AS BASES
- O SISTEMA BINÁRIO
- BITS E BYTES
- COMO O COMPUTADOR FAZ SOMAS

Os dez dedos das mãos levaram o homem a contar usando a base 10. Auxiliares de cálculo simples, como o ábaco, ou contador, são um modelo mecânico de nossos dedos.

não 10. Uma libra de peso, por exemplo, pode ser dividida em onças por sucessivas divisões ao meio.

CONTANDO COM NOVE DEDOS

Não há nenhuma razão que impeça o uso de um sistema baseado em qualquer número escolhido ao acaso. Se tivéssemos nascido, todos, com 1 dedo a menos em uma das mãos, provavelmente usaríamos um sistema ''nonário'' ou seja, de base 9. Contaríamos normalmente até 8 e, para obter o número seguinte, colocaríamos o 1 numa posição mais à esquerda e o 0 na posição à direita. Ou seja, o decimal 9 seria representado no sistema nonário pelo número 10.

Da mesma forma, se somássemos 1 a 18, teríamos 20; e 100 se somássemos 1 a 88.

Como se pode ver pelo exemplo acima, as regras da aritmética se aplicam sempre, não importando qual seja a fase do seu sistema de numeração. Você pode tentar operações simples num sistema de base 7 ou 8, e assim por diante. Qualquer que seja a base, as regras matemáticas são sempre válidas.

No nosso sistema nonário, a cada casa para a esquerda que o número desloca, seu valor absoluto é multiplicado por nove. Dessa forma, 3275 em nonário é igual a $5 + 7\times9 + 2\times9\times9 + 3\times9\times9\times9$, ou seja, 2417 em decimal. Você deve ter notado que não existe o número 9 no sistema nonário. Ele é representado por 10. Assim, seria correto dizer que, em nonário, 3275 é igual a $5 + 7\times10 + 2\times10\times10 + 3\times10\times10\times10$.

Com um pouco de agilidade mental você pode fazer alguns cálculos simples em nonário. Por exemplo: 99×9 dá 891 em decimal. Em nonário, o número 99 é representado por 120; isto é: $1\times9\times9 + 2\times9$, ou $81+18$. E 9 é 10.

Assim, em nonário, 99×9 se traduz por 120×10 ou 1200, que é $1\times9\times9\times9 + 2\times9\times9$, 891 em decimal.

Qualquer adição, subtração, multiplicação ou divisão de números nonários funcionará normalmente. Confira você mesmo. Só é preciso um pouco de cautela. Lembre-se que 16 em decimal é 17 em nonário.

PROBLEMAS DAS BASES ACIMA DE 10

Lidar com sistemas numéricos de base maior que 10 é um pouco mais difícil do que vimos até agora. Para manejá-los mais facilmente, temos que inventar novos números.

Dessa maneira, se você quisesse empregar um sistema de base 12, teria de inventar números para representar o 10 e o 11. Em duodecimal, 10 representa 1×12, ou 12. E 11 significa $1\times12 + 1$, ou seja, 13 em decimal.

O modo mais fácil de aumentar a faixa dos números é utilizar o alfabeto. O 10 pode ser A, e o 11, B. A2 seria, assim, 122; e BA, 142.

UM PROGRAMA PARA TODAS AS BASES

Fazer contas em bases de numeração que não sejam decimais pode ser bastante difícil e cansativo. Assim, apresentamos aqui um programa que faz contas em qualquer base até 36. Quando você rodá-lo, ele vai perguntar em que base quer trabalhar. Você deve responder em decimal.

A seguir, ele pede um número. Este deve ser digitado na base que você selecionou. Se foi a base 8, não deve haver nenhum número acima de 7. E se você escolheu uma base acima de 10, os números acima de 9 deverão ser digitados por meio das letras do alfabeto, sempre maiúsculas. 10 será A, 11, B, 12, C, e assim por diante.

Então, o computador perguntará que operação você deseja (+, −, * ou / representando respectivamente soma, subtração, multiplicação e divisão). Após isso, você deverá digitar outro número, na mesma base do anterior. Se a sua operação é divisão, e a resposta não é um inteiro, o computador lhe dirá que a operação não será realizada. ''Divisão não exata'', será a mensagem no vídeo. Se não, a operação e seu resultado aparecerão na tela na base que você escolheu para operar.

```
10 CLS
20 INPUT"BASE(ATE 36)";B
30 INPUT"NUMERO(INTEIRO) ";A$
40 INPUT"SINAL ";S$
50 INPUT"NUMERO(INTEIRO) ";B$
60 P$=A$:GOSUB 210
70 X$=STR$(DE)
80 P$=B$:GOSUB 210
```

```
90 Y$=STR$(DE)
100 IF S$="*" THEN X=VAL(X$)*VA
L(Y$)
110 IF S$="/" THEN X=VAL(X$)/VA
L(Y$)
120 IF S$="+" THEN X=VAL(X$)+VA
L(Y$)
130 IF S$="-" THEN X=VAL(X$)-VA
L(Y$)
140 IF X<>INT(X) THEN N$="IMPOS
SIVEL":GOTO 190
150 U=B*INT(X/B):IF X-U>9 THEN
N$=CHR$(55+(X-U))+N$:GOTO 170
160 N$=MID$(STR$(X-U)+N$,2)
170 X=INT(X/B)
180 IF X>0 GOTO 150
190 PRINT @257,A$+" "+S$+" "+B$
;"= ";N$
200 END
```

MICRO DICAS

UM MODELO PARA NÚMEROS EM DIFERENTES BASES

Às vezes é muito difícil visualizar como funciona um sistema de numeração de base diferente ao de base 10. Afinal, não é fácil ter que cortar fora alguns dedos, ou fazer crescer dedos adicionais nas mãos, de modo que você parece obrigado a se satisfazer com o que tem! Mas seria possível, por exemplo, grudar dois dedos um no outro com fita adesiva, para contar na base 9, ou usar dois palitinhos extras de contagem, para trabalhar na base 12.

Uma solução mais inteligente seria construir o seu próprio ábaco, ou calculadora de continhas, similar àqueles mostrados nas fotos que acompanham esta lição. Um ábaco convencional (mesmo aqueles de brinquedo, para criança) pode ajudar e agilizar as contas no sistema de base 10: assim, você pode criar um ábaco com tantas bolinhas em cada vareta da moldura quantas quiser. Não é necessário nem mesmo construir uma moldura cheia de arames, mas simplesmente arranjar uma série de sulcos paralelos, para conter as bolinhas e fazê-las rolar para cima ou para baixo.

Não se esqueça de que o número de continhas em cada fileira é um a menos do que a base em que você quer trabalhar. Assim, na base 10, você precisa de 9 contas, e na base 2, de apenas uma. Nenhuma bolinha equivale ao número zero, em qualquer sistema de numeração. Uma bolinha afastada das demais significa o número 1, e assim por diante. Quando a operação exigir mais de 9 bolinhas no sistema decimal, zere a fileira de volta, e suba uma bolinha na fileira seguinte (é a operação de "vai-um").

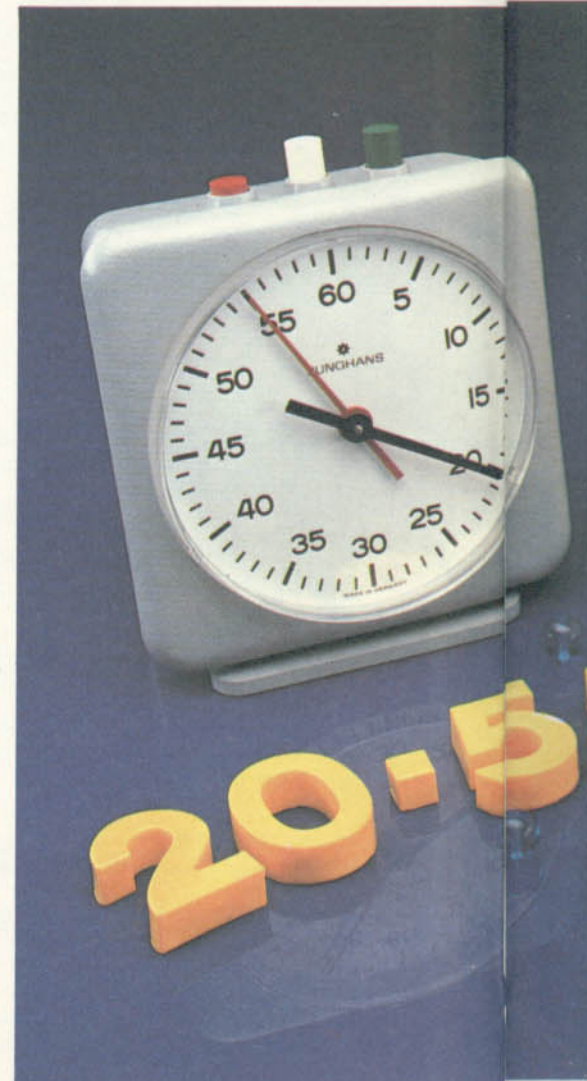
```
210 DE=0
220 IN=0
230 Y=ASC(RIGHT$(P$,1))
240 IF Y<58 THEN D=Y-48:GOTO 26
0
250 D=Y-55
260 DE=DE+D*B^IN
270 P$=LEFT$(P$,LEN(P$)-1)
280 IN=IN+1
290 IF LEN(P$)>0 THEN GOTO 230
300 RETURN
```

Este programa, como está, roda apenas nos compatíveis com o TRS-Color. Para rodá-lo nos modelos TRS-80, modifique a linha 190 para:

```
190 PRINT@513,A$+" "+S$ + " "
    + B$+ "= ";N$
```

```
10  POKE 23658,8
20 INPUT "Base (ate 36) ";b
30 INPUT "Numero (inteiro) ";
LINE a$
40 INPUT "Sinal "; LINE s$
50 INPUT "Numero (inteiro) ";
LINE b$
60 CLS
70 LET p$=a$: GOSUB 180
80 LET x$=STR$ dec
90 LET p$=b$: GOSUB 180
95 LET y$=STR$ dec
100 LET z$=x$+s$+y$: LET x=VAL
z$
110 IF x<>INT x THEN  LET n$="
Impossivel realizar a operacao
.": GOTO 160
120 LET n$=""
130 LET u=INT (x/b): LET u=u*b
: IF x-u>9 THEN  LET n$=CHR$ (
55+(x-u))+n$: GOTO 140
135 LET n$=STR$ (x-u)+n$
140 LET x=INT (x/b)
150 IF x>0 THEN  GOTO 130
160 PRINT AT 10,0;a$+s$+b$;"=
";n$
170 STOP
180 LET dec=0
190 LET indice=0
200 LET y=CODE p$(LEN p$)
210 IF y<58 THEN  LET d=y-48:
GOTO 220
215 LET d=y-55
220 LET dec=dec+d*b^indice
230 LET p$=p$(1 TO (LEN p$)-1)
240 LET indice=indice+1
250 IF LEN p$>0 THEN  GOTO 200
260 RETURN
```

MSX

```
10 CLS
20 INPUT "BASE (ATÉ 36)";B
30 INPUT "NÚMERO (INTEIRO) ";A$
40 INPUT "SINAL ";S$
50 INPUT "NÚMERO (INTEIRO) ";B$
60 P$=A$:GOSUB 210
70 X$=STR$(DE)
80 P$=B$:GOSUB 210
90 Y$=STR$(DE)
100 IF S$="*" THEN X=VAL(X$)*VA
L(Y$)
110 IF S$="/" THEN X=VAL(X$)/VA
L(Y$)
120 IF S$="+" THEN X=VAL(X$)+VA
L(Y$)
130 IF S$="-" THEN X=VAL(X$)-VA
L(Y$)
140 IF X<>INT(X) THEN N$="Opera
ção impossível":GOTO 190
150 U=B*INT(X/B):IF X-U>9 THEN
N$=CHR$(55+(X-U))+N$:GOTO 170
160 N$=MID$(STR$(X-U)+N$,2)
170 X=INT(X/B)
180 IF X>0 GOTO 150
190 LOCATE 0,12:PRINT A$+" "+S$
+" "+B$+" "+"="+" "+N$
200 END
210 DE=0
220 IN=0
230 Y=ASC(RIGHT$(P$,1))
240 IF Y<58 THEN D=Y-48:GOTO 26
0
250 D=Y-55
260 DE=DE+D*B^IN
270 P$=LEFT$(P$,LEN(P$)-1)
280 IN=IN+1
290 IF LEN(P$)>0 THEN GOTO 230
300 RETURN
```

```
10  HOME
20  INPUT "BASE (ATE 36) =>";B
30  INPUT "NUMERO (INTEIRO) =>"
;A$
40  INPUT "OPERACAO (+,-,*,/) =
>";S$
50  INPUT "NUMERO (INTEIRO) =>"
;B$
60 P$ = A$: GOSUB 210
70 X = DE
80 P$ = B$: GOSUB 210
90 Y = DE
100  IF S$ = "*" THEN X = X * Y
110  IF S$ = "/" THEN X = X / Y
120  IF S$ = "+" THEN X = X + Y
130  IF S$ = "-" THEN X = X - Y
140  IF X <  >  INT (X) THEN N$
 = "NAO DA": GOTO 190
150 U = B *  INT (X / B): IF X
- U > 9 THEN N$ =  CHR$ (55 + (
X - U)) + N$: GOTO 170
160 N$ =  STR$ (X - U) + N$
170 X =  INT (X / B)
180  IF X > 0 THEN 150
190  HTAB 10: VTAB 15: PRINT A$
;" ";S$;" ";B$;" = ";N$
200  END
210 DE = 0
220 IN = 0
230 Y =  ASC ( RIGHT$ (P$,1))
240  IF Y < 58 THEN D = Y - 48:
 GOTO 260
250 D = Y - 55
260 DE = DE + D * B ^ IN
265  IF  LEN (P$) = 1 THEN  RET
URN
270 P$ =  LEFT$ (P$, LEN (P$) -
 1)
280 IN = IN + 1
290  GOTO 230
300  RETURN
```

Se você acha que fazer contas em diversas bases é mais do que pretende saber algum dia, note que o computador também trapaceia. O programa, na realidade, converte os números que você digitou em decimais, executa a operação e reconverte o resultado para a base que você escolheu. Ao contrário da mente humana, o computador só consegue fazer cálculos em decimal, dentro de um programa escrito em BASIC, apesar de converter os números para binário, quando as instruções são traduzidas para código de máquina durante a execução do programa.

E AGORA, BINÁRIOS

Para os computadores digitais, o sistema de numeração mais apropriado é o baseado no número 2. Isso acontece porque o computador é composto de circuitos eletrônicos que possuem dois estados evidentes: ligado e desligado. O estado "desligado" representa o número 0, e o "ligado", o número 1. E, na base 2, estes são todos os números de que você necessita para expressar qualquer valor.

O sistema de numeração de base 2 é conhecido como "binário". Ele é composto apenas por zeros e uns; todos os outros números foram abolidos. Dessa forma, se você começar a contar da maneira habitual, não vai passar de 1. Some 1 a 1 e terá o número 1 deslocado uma casa para a esquerda, enquanto a casa da direita é zerada. Assim, em binário, $1+1 = 10$.

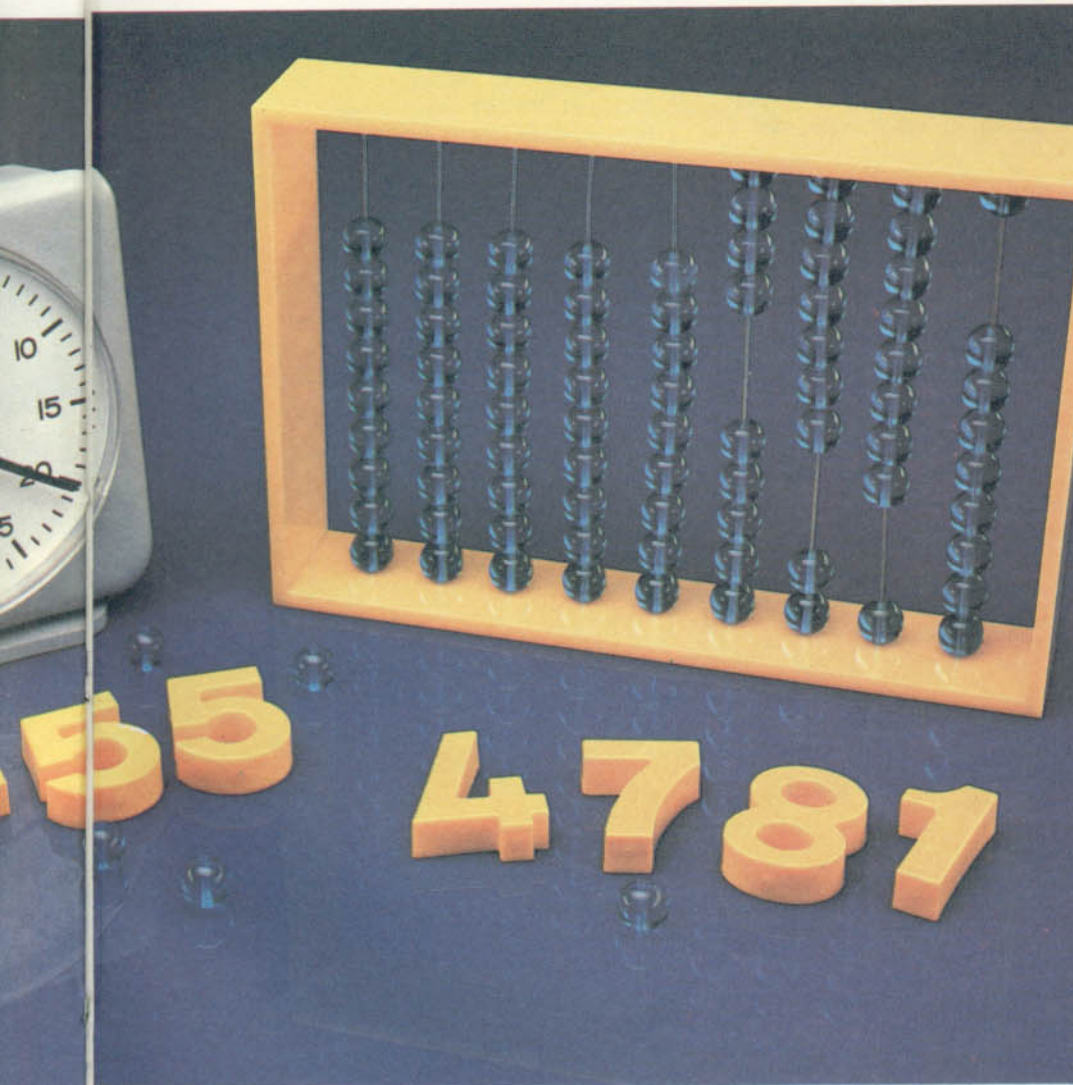
Contar de 0 a 8 resulta em 0, 1, 10, 11, 100, 101, 110, 111, 1000. Novamente, estes números obedecem às leis da aritmética. Se você quiser somar $10+11$, deve primeiramente somar os dois números da direita, $0+1$. A seguir, devem ser somados os dois da esquerda, $1+1$, o que dá 2; só que 2 em binário é 10. Assim $10+11 = 101$; ou seja, $2+3 = 5$.

Subtração em binário é um processo tão direto quanto a soma. O único detalhe a observar é o número que se "leva" para a próxima casa.

A multiplicação e a divisão são extraordinariamente fáceis, mesmo para seres humanos "normais". Na realidade, é tão fácil que "até" um computador pode fazer essas operações. No primeiro caso, tudo se resume a multiplicar 0×0, que dá 0; 0×1, que também dá 0; ou 1×1, que dá 1.

A divisão é igualmente simples: em cada operação, basta decidir se o divisor está dentro do dividendo uma vez ou nenhuma vez.

Confira você mesmo como os processos aritméticos funcionam em binário. O painel das páginas seguintes dá exemplos de operações para você fazer.

Qualquer sistema de numeração pode ser imitado por um modelo mecânico: o relógio, por exemplo, usa a base 60, enquanto o ábaco utiliza a base 10.

BITS E BYTES

Os números binários refletem exatamente o que acontece no computador. As instruções ou dados que você dá a ele são codificados em números binários, que, por sua vez, são manipulados e armazenados pelos circuitos internos do aparelho. Assim, quando você começa a dominar o sistema binário, começa também a compreender como funciona o computador.

Cada um dos dígitos de um número binário é representado eletronicamente dentro do computador por um circuito ligado ou desligado. Se está ligado, o valor é 1. Se, pelo contrário, o circuito está desligado, o valor é 0. Cada dígito é conhecido como um *bit*.

Esses circuitos são agrupados em unidades maiores, que representam números também maiores e mais úteis. Em quase todos os micros, os bits estão organizados em grupos de oito, formando o que se conhece como um *byte*. Cada byte representa, assim, oito dígitos binários, de forma que pode armazenar qualquer número entre 00000000, ou zero, e 11111111, ou 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1, que é 255. Números maiores que 255 são representados por dois ou mais bytes.

FRAÇÕES BINÁRIAS

Todos os programas e exemplos dados até aqui manipularam números inteiros. É possível, também, converter frações em números binários.

Por exemplo, 1/2 é 0.1 em binário; 1/4 é 0.01 e 1/8 é 0.001. Existe um padrão determinado. Qualquer fração pode ser formada a partir de 1/2s, 1/4s, 1/8s, 1/16s, etc., da mesma forma que qualquer número inteiro pode ser formado a partir de 1s, 2s, 4s, 8s, etc. O problema é que é mais difícil trabalhar com frações binárias porque sempre temos uma infinidade de 0s e 1s!

Quando você converte frações, é melhor pensar nelas como 0.5; 0,25 e assim por diante. Desta forma, você pode manipular uma fração como 0.75 como sendo $1 \times 0.5 + 1 \times 0.25$, o que significa em binário 0.11.

O próximo programa permite que você trabalhe com frações binárias sem grandes dificuldades:

```
10 CLS
20 PRINT"CONVERSAO DE DECIMAL P
/ BINARIO"
30 PRINT:INPUT"DIGITE UM NUMERO
 ENTRE 0 E 1    ";N
40 IF N<=0 OR N>=1 THEN 30
50 N$="0."
60 FOR T=1 TO 30
70 N=N*2
80 N$=N$+CHR$(48+INT(N))
90 N=N-INT(N)
100 NEXT T
110 PRINT @257,"O NUMERO BINARI
O E: "
120 PRINT:PRINT N$:PRINT
130 PRINT"OUTRO NUMERO (S/N) ?"
140 A$=INKEY$
150 IF A$="S" THEN 10
160 IF A$<>"N" THEN 140
170 END
```

O programa acima, como está, roda apenas nos compatíveis com o TRS-Color. Para os modelos tipo TRS-80, modifique a linha 110 para:

```
110 PRINT@513,"O NUMERO BINARIO
    E' :-"
```

SS

```
10 CLS
20 PRINT '"Conversao de Decim
al para Binario"
30 INPUT "Digite um numero en
tre 0 e 1",N
40 IF N<=0 OR N>=1 THEN  GOTO
30
45 PRINT ''"Decimal: ";N
50 LET N$="0."
60 FOR T=1 TO 16
70 LET N=N*2+1E-9
80 LET N$=N$+CHR$ (48+INT N)
90 LET N=N-INT N
100 NEXT T
110 PRINT ''"O numero binario
e:"
120 PRINT N$
130 PRINT '''''"Outro numero (
S/N)?"
140 LET A$=INKEY$
150 IF A$="S" OR A$="s" THEN
GOTO 10
160 IF A$<>"N" AND A$<>"n"
THEN  GOTO 140
```

Tal como está, o programa acima roda apenas no Sinclair Spectrum

```
ADIÇÃO BINARIA              DECIMAL

 1100111              1100111 = 103

 +100010               100010 =  34
 -------
10001001             103 + 34 = 137

                     10001001 = 137
```

A adição binária funciona a partir da direita. 0 + 0 dá 0, 0 + 1 dá 1, e 1 + 1 resulta em 10, que é o 0 com "vai-um" para o próximo dígito. Observe como essas operações funcionam na soma realizada acima.

```
SUBTRAÇÃO BINARIA           DECIMAL

 1011001              1011001 =  89

 -110011               110011 =  51
--------
  100110              89 - 51 =  38

                       100110 =  38
```

A subtração binária é igualmente simples. 1 − 1 dá 0, 0 − 1 dá 1. Agora, a operação 1 − 0 dá 0 e "vai-um" para o próximo dígito à esquerda. Os resultados, portanto, não são todos exatamente iguais ao da mesma operação no sistema decimal.

(TK-90X). Para rodar no ZX-81, substitua as linhas seguintes:

```
30 PRINT "ENTRE UM NUMERO ENTRE
   0 E 1"
35 INPUT N
80 LET N$=N$+CHR$(28+INT N)
```

```
10 CLS
20 PRINT TAB(7);"BINÁRIO >> DEC
IMAL"
30 PRINT:INPUT " digite um núme
ro entre 0 e 1 ";N
40 IF N<=0 OR N>=1 THEN 30
50 N$="0."
60 FOR T=1 TO 30
70 N=N*2
80 N$=N$+CHR$(48+INT(N))
90 N=N-INT(N)
100 NEXT T
110 LOCATE 0,10:PRINT "O número
 binário é : "
120 PRINT:PRINT N$:PRINT
130 PRINT "outro número (S/N) ?
"
140 A$=INKEY$
150 IF A$="S" THEN 10
160 IF A$<>"N" THEN 140
170 END
```

```
10  HOME
20  HTAB 5: PRINT "CONVERSAO DE
```

Um ábaco binário utiliza uma conta em cada vareta. No computador, essa disposição é substituída por pulsos elétricos.

```
MULTIPLICACAO BINARIA       DECIMAL

        1101                   1101 =  13
x       1011                   1011 =  11
------------
        1101                13 x 11 = 143
       1101                10001111 = 143
      0000
     1101
------------
    10001111
```

```
DIVISAO BINARIA                DECIMAL
       |
 11001 | 101                   11001 =  25
       L---
 101     101                     101 =   5
  000                         25 / 5 =   5
   101                           101 =   5
```

Em cada fase da multiplicação binária, você multiplica 0 por 0, que dá 0, ou 1 por 0, que também dá 0, ou ainda 1 por 1, que resulta sempre em 1. Veja como isto se aplica no exemplo acima, onde é simulada uma operação no vídeo do computador.

Na divisão binária, finalmente, é necessário verificar se o divisor cabe no dividendo pelo menos uma vez. Se isto acontecer, o resultado será 1; caso contrário, será 0. Isso faz da divisão uma das operações binárias mais simples.

```
CIMAL PARA BINARIO"
30  PRINT : PRINT "DIGITE UM NU
MERO": INPUT "ENTRE 0 E 1 =>";N
40  IF N <  = 0 OR N >  = 1 THE
N 30
50 N$ = "0."
60  FOR T = 1 TO 38
70 N = N * 2
80 N$ = N$ +  CHR$ (48 +  INT (
N))
90 N = N -  INT (N)
100  NEXT T
110  HTAB 1: VTAB 15: PRINT "O
NUMERO BINARIO E: "
120  PRINT : PRINT N$: PRINT
130  PRINT : PRINT "OUTRO NUMER
O? ";
140  GET A$
150  IF A$ = "S" THEN 10
160  IF A$ <  > "N" THEN 130
170  END
```

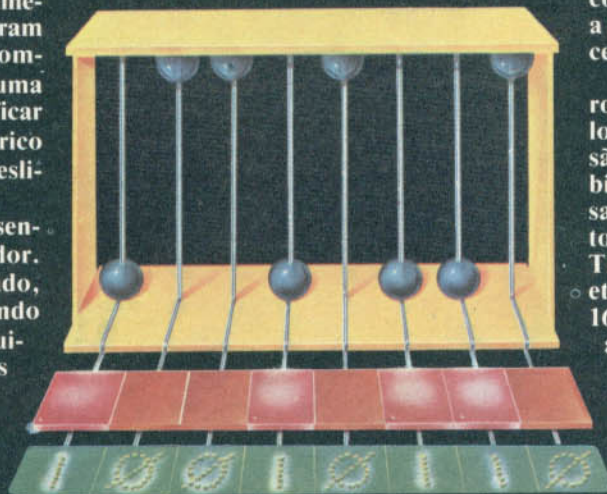
O programa aceita qualquer número entre 0 e 1 e imprime o seu equivalente binário. A linha 70 começa por dobrar o seu número e a linha 80 constrói o número binário. Inicialmente o computador trabalha com o valor de INT(N), que será ou 0 ou 1. Então, a esse valor adiciona-se 48 para perfazer 48 ou 49, que são os códigos ASCII para 0 e 1. Tudo isto é feito de forma a transformar os números em cadeias de caracteres para que possam ser adicionados à variável **N$**. Na linha 90 tiramos a parte inteira de N para deixar apenas a decimal.

O programa no Sinclair Spectrum é um pouco diferente, porque enquanto INT 1 é 1, como se espera, INT (.5 × 2) é 0. Isto acontece porque ele armazena .5 × 2 como .99999999... Assim, uma pequena fração tem que ser adicionada a N na linha 70 para fazer com que INT funcione como deve.

Nos modelos compatíveis com o ZX-81, a linha 80 usa outro número para somar ao valor inteiro de N (observe que essa máquina não usa o código ASCII). O valor 28 corresponde ao caractere 0, o valor 29 ao caractere 1, etc. e assim sucessivamente.

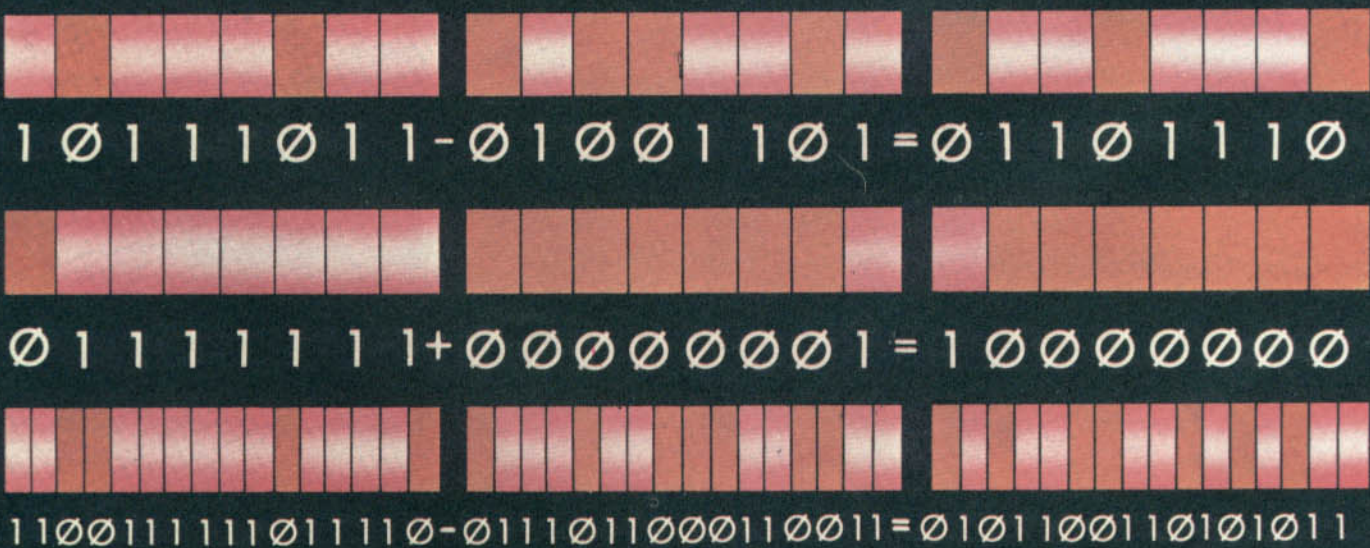
O LONGO TRAJETO DO ÁBACO À ELETRÔNICA

Os números binários podem ser adicionados, subtraídos, multiplicados e divididos como qualquer número na base 10. Entretanto, eles foram escolhidos para utilização nos computadores digitais em virtude de uma propriedade singular: é fácil verificar se um determinado circuito elétrico ou eletrônico está ligado ou desligado.

Os dígitos binários são representados por circuitos de computador. Quando um circuito está desligado, ele representa o dígito 0, e quando está ligado, o dígito 1. Nos circuitos integrados do computador, os circuitos binários são arranjados em grupos de 8, de tal forma que valores binários de 8 bits podem ser representados. Isso dá uma gama de representação de 00000000 — ou 0 em decimal — a 11111111 — ou 255 em decimal. No manual do seu computador esse "número mágico" — 255 (ou 256, se você contar a partir de 1, ao invés de 0) — aparece a todo momento.

Às vezes são necessários números binários muito maiores, e duas locações de memória de 8 bits cada são juntadas para obter um número binário de 16 bits. O microprocessador Zilog Z-80, existente em muitos computadores pessoais, como o TRS-80, os compatíveis com MSX, etc., pode trabalhar com números de 16 bits, e até mesmo fazer operações aritméticas com eles. Nestes casos, números binários entre 0000000000000000 e 1111111111111111, ou seja, entre 0 e 65.535, em decimal, podem ser manipulados.

1 Ø 1 1 1 Ø 1 1 - Ø 1 Ø Ø 1 1 Ø 1 = Ø 1 1 Ø 1 1 1 Ø

Ø 1 1 1 1 1 1 1 + Ø Ø Ø Ø Ø Ø Ø 1 = 1 Ø Ø Ø Ø Ø Ø Ø

11ØØ1111 11Ø1111Ø - Ø111Ø11ØØØ11ØØ11 = Ø1Ø1 1ØØ1 1Ø1Ø1Ø11

ENSINE SEU MICRO A TOMAR DECISÕES

- ESCOLHA O CAMINHO A SEGUIR
- DECISÕES MAIS COMPLICADAS
- APRENDA A UTILIZAR O COMANDO **IF...THEN...ELSE**

Embora não tenha cérebro, o computador tem capacidade para decidir de forma lógica. Aprenda a trabalhar com a instrução IF...THEN e ensine seu micro a tomar decisões corretamente.

O que torna um computador infinitamente superior a uma máquina de calcular é sua capacidade de tomar decisões baseadas em raciocínios lógicos. Um desses raciocínios é o comando **IF...THEN**. O computador reage a essa declaração de modo semelhante a um ser humano: **SE (IF)** isto é verdadeiro, **ENTÃO (THEN)** faça tal coisa. Eis um exemplo:

IF A<18 (ou seja, SE A é menor que 18) **THEN PRINT** "menor de idade" (ENTÃO IMPRIMA "menor de idade")

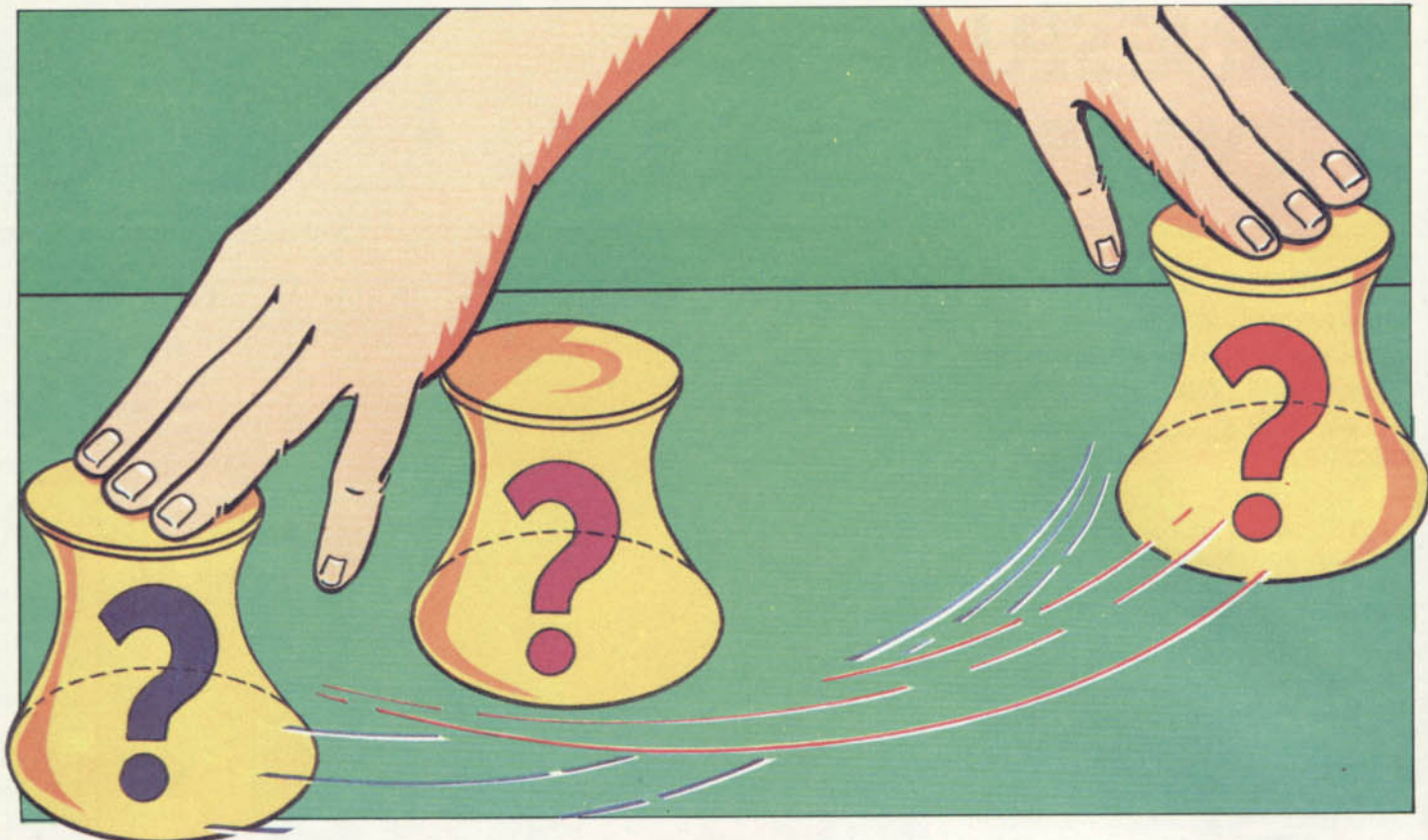
Ao encontrar a declaração **IF**, o computador verifica se a proposição que a segue é verdadeira. Se for, ele executa tudo o que estiver após o **THEN**. Se a proposição é falsa, ele ignora o resto e passa para a próxima linha do programa.

Você verá como funciona tudo isso no programa a seguir.

```
10 PRINT "Digite a lista das no
tas"
20 PRINT "Digite -99 para termi
nar"
30 INPUT N
40 IF N=-99 THEN PRINT "Média =
 ";T/C:STOP
50 LET T=T+N
60 LET C=C+1
70 GOTO 30
```

A mensagem no início do programa instrui o usuário a digitar primeiro uma lista de notas e por fim o número -99. As linhas 25 e 26 zeram o valor das variáveis do total (**T**) e do contador (**C**). A linha 30 dá entrada ao valor que você digita. A linha 50 adiciona esse valor ao total corrente, e a linha 60 conta quantos números você digitou, adicionando 1 à variável do contador cada vez que um novo número é dado.

Enquanto você estiver digitando no-

tas reais, o computador ignorará a linha 40 e seguirá em frente, mas, quando você digitar -99, a condição da linha 40 N = -99 será satisfeita, e o computador imprimirá a média das notas (**T/C**). Números como -99 são chamados de fictícios, ou terminais, e são úteis para controlar o que acontece em um programa. Este é um recurso muito usado para interromper partes de um programa.

UMA ESCOLHA COM TRÊS CAMINHOS

E se você quiser escolher entre três ou mais alternativas, de modo a colocar o computador em diferentes cursos de ação? É simples: o computador sorteia um número que você tem de adivinhar; ao comparar o seu palpite com o número sorteado, ele elabora três alternativas: o palpite foi correto, foi muito baixo, ou muito alto. Tudo se passa, portanto, como num jogo de advinhação.

```
5 CLS
10 LET N=RND(20)
20 PRINT "ACABO DE IMAGINAR UM
NUMERO..."
30 PRINT "TENTE ADIVINHA-LO!"
40 INPUT G
50 IF G=N THEN PRINT "MUITO BEM
!":FOR D=1 TO 2000:NEXT D:GOTO
10
60 IF G<N THEN PRINT "ESTA BAIX
O, TENTE OUTRA VEZ!"
70 IF G>N THEN PRINT "MUITO ALT
O! TENTE DE NOVO."
80 GOTO 40
```

Atenção: digite o programa abaixo em maiúsculas, se quiser executá-lo nos compatíveis com ZX-81:

```
 5 CLS
10 LET N=INT (RND*20)+1
20 PRINT "Eu acabo de imagina
r um numero."
30 PRINT "Tente adivinha-lo."
40 INPUT G
50 IF G=N THEN  PRINT "Muito
bem!": PAUSE 100: GOTO 10
60 IF G<N THEN  PRINT "Esta b
aixo; tente outra vez!"
70 IF G>N THEN  PRINT "Muito
alto! Tente de novo."
80 GOTO 40
```

O programa abaixo, como está, rodará apenas nos micros tipo MSX. Para rodá-lo no Apple II e no TK-2000, elimine a linha 3 e substitua o comando da linha 5 por **HOME**.

```
3 R=RND(-TIME)
5 CLS
10 LET N=INT(RND(1)*21)
20 PRINT "Acabo de pensar e um
número"
30 PRINT "...será que você cons
egue advinhá-lo ?"
40 INPUT G
50 IF G=N THEN PRINT "Certo, mu
ito bem!":FOR D=1 TO 2000:NEXT
D:GOTO 5
60 IF G<N THEN PRINT "Muito bai
xo, tente outra vez."
70 IF G>N THEN PRINT "Alto dema
is, tente outra vez."
80 GOTO 40
```

A linha 10 escolhe um número ao acaso entre 1 e 20, e as linhas 20 a 40 convidam você a adivinhá-lo. Qualquer que seja sua resposta, o computador verifica, nas linhas 50, 60 e 70, que proposição é verdadeira.

Suponha que seu palpite seja um número abaixo do sorteado. Neste caso, o computador vai primeiro para a linha 50; mas P = N é falso, de modo que ele ignora o resto da linha e passa para a

seguinte, a linha 60. Agora, a proposição é verdadeira, visto que P<N. Então, ele imprime a mensagem "Muito baixo, tente novamente". A seguir, ele passa assim mesmo para a próxima linha, onde a proposição é falsa, de forma que ele a ignora e vai para a 80, que leva você de volta para outra tentativa.

Este programa funciona muito bem, mas tem uma desvantagem: ele continua a pedir um novo palpite ao jogador, sem se importar se este quer ou não continuar jogando. Seria interessante fazer com que o computador preguntasse se você quer fazer uma nova tentativa.

As próximas linhas fazem exatamente isso. O comando **IF...THEN** será novamente utilizado, só que desta vez o computador estará comparando letras e não números.

Antes de entrá-las em adição ao programa anterior, não esqueça de mudar a linha 50 para:

```
50 IF G=N THEN PRINT "CERTO,
   MUITO BEM!":GOTO 100
```

```
100 PRINT "QUER JOGAR DE NOVO?(
S/N)"
110 LET A$=INKEY$:IF A$=""THEN
GOTO 110
120 IF A$="S" THEN RUN
```

```
100 PRINT "Voce quer tentar ou
tra vez(S/N)?"
110 LET A$=INKEY$: IF A$<>"S"
AND A$<>"N" THEN  GOTO 110
120 IF A$="S" THEN  RUN
```

Atenção: para poder rodar o programa abaixo em computadores tipo Apple e TK-2000, substitua a linha

```
110 GET A$
```

```
100 PRINT ,,"Quer jogar outra v
ez (S/N)?"
110 LET A$=INKEY$: IF A$="" THE
N GOTO 110
120 IF A$="S" OR A$="s" THEN GO
TO 5
```

A linha 110 espera que você acione uma tecla. Caso seja o "S" maiúsculo, o programa continuará. Se qualquer outra tecla for acionada, o programa terminará.

CHECAGEM DUPLA

Para fazer o computador testar duas ou mais condições, antes de decidir que caminho tomar, use palavras especiais chamadas *operadores lógicos:*

```
100 IF D$="DOMINGO" AND H=20
    THEN PRINT "E' HORA DO
    FANTASTICO"
```

Quando você usa **AND** (a conjunção *e*, em inglês) entre duas condições, ambas devem ser verdadeiras para que o computador execute o resto da linha. Se uma delas ou as duas forem falsas, ele passará para a próxima linha. Nesse exemplo é necessário que sejam 20 horas de um domingo para que o computador imprima mensagem. Veja outro exemplo:

```
200 IF P$="SAGU" OR
    P$="TAPIOCA" THEN PRINT
    "NAO ESTOU COM FOME"
```

Essa linha usa **OR** (*ou*, em inglês) e o computador imprimirá a mensagem no caso de pelo menos uma das comparações ser verdadeira.

O teste pode vir a ser muito complicado se houver uma quantidade muito grande de condições para verificar. Se você tem vários AND e OR juntos, terá que usar parênteses para que o computador saiba o que verificar primeiro.

Uma linha de um jogo de aventuras pode se parecer com isto:

```
2000 IF P=14 AND (C$="ESPADA"
     OR C$="FACA") THEN PRINT
     "VOCE DESTRUIU O GREMLIN"
```

Isso só é verdadeiro se você estiver na posição 14 E com uma espada OU uma faca. Mude os parênteses:

```
2000 IF (P=14 AND C$="ESPADA")
     OR C$="FACA" THEN PRINT
     "VOCE DESTRUIU O GREMLIN"
```

Agora ela será verdadeira se você estiver na posição 14 com uma espada OU se estiver em qualquer lugar com uma faca.

TIRE A SORTE GRANDE

Agora, usando os conceitos que aprendemos, vamos programar um joguinho que simula uma máquina caça-níqueis. Ela faz um bom uso de **AND** e **OR**. Tente tirar a sorte grande!

```
20 LET M=50
30 CLS
40 LET M=M-5
50 IF M<0 THEN PRINT "DESCULPE.
.. VOCE ESTA DURO!":END
60 LET A=RND(12)+192
70 LET B=RND(12)+192
80 LET C=RND(12)+192
210 PRINT @237,CHR$(A):PRINT@23
9,CHR$(B):PRINT @241,CHR$(C)
220 IF A=B AND B=C THEN PRINT @
258,"VOCE GANHOU $50!!!":LET M=
M+50
230 IF (A=B  OR B=C) AND A<>C T
HEN PRINT @361,"VOCE GANHOU $10
!":LET M=M+10
240 FOR D=1 TO 500:NEXT
250 PRINT @323,"QUER JOGAR OUTR
A VEZ?(S/N)":PRINT @361,"VOCE T
EM $";M
```

OS OPERADORES LÓGICOS

Se você não está acostumado com os símbolos usados em matemática para expressar relações lógicas, como "maior que" e "menor que" — os chamados *operadores* — pode achá-los meio confusos, no início.

Assim, pense neles como uma espécie de cunha. No símbolo > o lado mais aberto, à esquerda, é *maior do que* o lado mais agudo (a ponta). No símbolo < o lado mais estreito, á esquerda, é *menor do que* o lado direito. Deste modo, a expressão A>B deve ser lida simplesmente: A é maior do que B. Colocar o sinal de igualdade (=) ao lado de um dos operadores prévios significa também que pode ser "maior ou igual a" ou "menor ou igual a".

Eis aqui a lista de todas as combinações possíveis:

A=B : A é igual a B
A>B : A é maior que B
A<B : A é menor que B
A>=B : A é maior ou igual a B
A<=B : A é menor ou igual a B
A<>B : A é diferente de B

Alguns computadores exigem que a ordem de digitação dos operadores compostos seja sempre a mesma. Por exemplo, não se pode digitar >= ou => indistintamente: eles não vão aceitar. Em outros micros, o esquema é mais liberal.

Nos computadores da linha Sinclair os operadores compostos são digitados através de uma única tecla. Dessa forma, se você tentar entrar o sinal < seguido do sinal =, por exemplo, a linha gera um erro de sintaxe.

Finalmente, alguns computadores aceitam também o sinal # (sustenido), para significar "diferente de".

Posso combinar dois ou mais IF...THEN em uma única linha?

Sim, mas esta não é uma idéia muito boa, pois dificulta muito a compreensão de como funciona um programa e pode gerar erros lógicos.

O princípio básico da declaração **IF...THEN** é que o que vem depois do **THEN** na linha do **IF**, só será executado se a proposição for verdadeira. Assim, na linha:

```
70 IF X=Y THEN PRINT "NAVE
   DESTRUIDA"  :  LET NAVES=
   NAVES-1:GOTO 30
```

Nenhuma das instruções após o **THEN** será executada, se X for diferente de Y. Mas, algumas vezes o **IF** composto pode ser útil:

```
70 IF X=Y THEN PRINT "NAVE
   DESTRUIDA": IF NAVES>0 THEN
   LET NAVES=NAVES-1

80 IF X=Y AND NAVES=0 THEN
   PRINT "FIM DO JOGO":END
```

Note como os dois **IFs** na linha 70 simplificam a programação.

Por que ocorrem erros quando tento rodar programas digitados?

Pode ser que existam falhas no programa em si, mas os erros mais comuns são de digitação, quando se está copiando. Eis alguns deles (lembre-se de checá-los um a um, quando estiver copiando programas):

— Confundir a letra maiúscula I ou a letra L minúscula com o número 1; ou ainda a letra O maiúscula com o zero.

— Confundir parênteses com os sinais de maior (>) e menor (<). Confundir o cifrão ($) com a letra S, ou omiti-los.

— Omitir as aspas no começo ou no fim, em comando **PRINT** ou **LET**. Omitir os dois pontos (:) entre instruções na mesma linha.

— Esquecer a vírgula entre dois ítens de uma declaração **DATA**, o que pode determinar um número muito grande para o computador, ou a falta de um item quando o computador chega ao final dos **DATA**.

— Omitir um sinal de menos (em um programa que gera gráficos na tela, isso pode resultar em uma tentativa de desenhar além dos seus limites).

— Omitir um sinal de ponto e vírgula no final de uma linha **PRINT** ou colocar um número de espaços em branco maior ou menor do que o previsto.

```
260 LET K$=INKEY$:IF K$="" THEN
 GOTO 260
270 IF K$="S" THEN GOTO 30
280 END
```

```
 20 LET M=50
 30 CLS
 40 LET M=M-5
 50 IF M<0 THEN  PRINT "Descul
pe... Voce esta duro!": STOP
 60 LET A=INT (RND*12)+130
 70 LET B=INT (RND*12)+130
 80 LET C=INT (RND*12)+130
210 PRINT  PAPER 0; INK 4;AT
10,14;CHR$ A;AT 10,16;CHR$ B;
AT 10,18;CHR$ C
220 IF A=B AND B=C THEN
PRINT AT 13,2;"Voce ganhou $50
!!!": LET M=M+50
230 IF (A=B OR B=C) AND A<>C
THEN  PRINT AT 13,9;"Voce ganh
ou $10!": LET M=M+10
240 PAUSE 25
250 PRINT AT 15,6;"Quer jogar
outra vez?(S/N)": PRINT TAB 10
;"Voce tem $";M
260 IF INKEY$="" THEN  GOTO
260
270 IF INKEY$="n" THEN  STOP
280 GOTO 30
```

```
20 LET M=50
30 CLS
40 M=M-5
50 IF M<0 THEN PRINT "Desculpe,
 seu dinheiro acabou.":END
60 LET A=INT(RND(1)*5+66)
70 LET B=INT(RND(1)*5+66)
80 LET C=INT(RND(1)*5+66)
90 LOCATE 17,10:PRINT CHR$(1)+C
HR$(A);" ";CHR$(1)+CHR$(B);" ";
CHR$(1)+CHR$(C)
220 IF A=B AND B=C THEN LOCATE
2,16:PRINT " Você ganhou o prêm
io máximo ... $50":LET M=M+50
230 IF (A=B OR B=C) AND A<>C TH
EN LOCATE 10,16:PRINT "Você gan
hou $10":LET M=M+10
240 FOR D=1 TO 500:NEXT
250 LOCATE 5,18:PRINT " Quer te
ntar outra vez ? (S/N)"
260 LOCATE 12,20:PRINT "Você te
m $";M;"    ";
270 LET K$=INKEY$:IF K$="" THEN
 GOTO 270
280 IF K$="s" OR K$="S" THEN GO
TO 30
290 END
```

Como o Apple II não tem caracteres gráficos pré-definidos, a versão a seguir faz na tela blocos coloridos (símbolos do caça-níqueis):

```
20  LET M = 50
30  HOME : GR
40  LET M = M - 5
50  IF M < 0 THEN  VTAB 22: PRI
NT "DESCULPE, VOCE ESTA DURO":
END
60  LET A =  INT ( RND (1) * 12
 + 4)
70  LET B =  INT ( RND (1) * 12
 + 4)
80  LET C =  INT ( RND (1) * 12
 + 4)
85  COLOR= A: VLIN 0,39 AT 16
90  COLOR= B: VLIN 0,39 AT 18
95  COLOR= C: VLIN 0,39 AT 20
110  IF A = B AND B = C THEN  V
TAB 22: PRINT "VOCE  GANHOU $50
": LET M = M + 50
120  IF (A = B OR B = C) AND A
<  > C THEN  VTAB 22: PRINT "VO
CE GANHOU $10": LET M = M + 10
130  FOR I = 1 TO 500: NEXT I
140  VTAB 22: PRINT "OUTRA VEZ?
 (S/N)?": PRINT "VOCE TEM $";M
150  GET A$
160  IF A$ = "N" THEN  END
170  GOTO 30
```

A versão para o TK-2000 usa os caracteres do teclado. Só as linhas que serão substituídas no programa acima aparecem abaixo:

```
30  HOME
60  LET A =  INT ( RND (1) * 12
) + 222
70  LET B =  INT ( RND (1) * 12
) + 222
80  LET C =  INT ( RND (1) * 12
) + 222
90  VTAB 10
100  PRINT  TAB( 18); CHR$ (242
); CHR$ (A); TAB( 209; CHR$ (24
2); CHR$ (B); TAB( 22); CHR$ (2
42); CHR$ (C)
```

O programa usa várias linhas **IF...THEN**. A primeira, de número 50, checa se você tem dinheiro para jogar. Se você tem, a linha é ignorada, mas, se não, uma mensagem é mostrada, e o jogo termina.

As linhas 60 a 80 escolhem três números aleatórios, e a linha 210 converte esses números em caracteres (no caso dos computadores das linhas TRS-Color, Spectrum e MSX, naipes de baralho) e os coloca no centro da tela. O Sinclair e o TRS-Color convertem esses números em caracteres. Os outros necessitam de ajustes.

Na linha 220, se os três caracteres são iguais, você ganha o *jack pot* (maior prêmio) e seu dinheiro é aumentado em $50. Na linha 230 você ganha $10 se dois caracteres adjacentes são iguais (A = B ou B = C), mas não ganha nada se os dois caracteres das extremidades são iguais. Se você não ganhar nada, o computador ignora as linhas 220 e 230 e pas-

sa para a 240, que provoca uma breve pausa.

As linhas seguintes são uma outra versão da rotina "sim/não" que checam se você quer uma nova jogada.

IF...THEN...ELSE

Em alguns computadores você pode escrever a instrução **IF** de uma forma muito mais poderosa, chamada de **IF...THEN...ELSE**. Veja um exemplo:

```
10 INPUT IDADE
20 IF IDADE<18 THEN PRINT
   "MENOR DE IDADE" ELSE PRINT
   "ELEGIVEL"
```

Esta linha faz exatamente o que diz. Se você tem menos de 18 anos, o computador imprime "MENOR DE IDADE", mas, se você tiver 18 ou mais, ele imprimirá "ELEGÍVEL"

Dos computadores que apresentam a declaração **IF...THEN...ELSE** em seu repertório de BASIC apenas os das linhas TRS-80 e MSX estão ilustrados aqui.

Com a instrução **IF...THEN...ELSE** o programa fica mais fácil de ler e escrever. No entanto, ela não é essencial. Você pode escrever programas sem ela se o seu computador usa apenas **IF...THEN**. Existem duas maneiras de contornar o problema. Uma emprega **IF...THEN** seguido de **GOTO** para pular para a parte requerida do programa:

```
10 INPUT IDADE
20 IF IDADE<18 THEN PRINT
   "MENOR DE IDADE":GOTO 30
25 PRINT "ELEGIVEL"
30 ...restante do programa
```

Esse tipo de programação serve para os micros que permitem *comandos múltiplos por linha*, isto é, a colocação de mais de uma instrução na linha do **IF**. Os compatíveis com o ZX-81 não contam com essa possibilidade. A segunda solução do problema apóia-se num **IF...THEN** extra para assegurar que todas as possibilidades sejam cobertas:

```
10 INPUT IDADE
20 IF IDADE<18 THEN PRINT
   "MENOR DE IDADE"
25 IF IDADE >=18 THEN PRINT
   "ELEGIVEL"
30 ...restante do programa
```

Como dois **IFs** demoram mais tempo do que um só para serem executados (e isso é importante quando o mesmo trecho de programa deve ser executado muitas vezes), outro estilo de programação de alternativas é:

```
10 INPUT IDADE
20 IF IDADE>=18 THEN GOTO 25
22 PRINT "MENOR DE IDADE"
24 GOTO 30
25 PRINT "ELEGIVEL"
30...resto do programa
```

Nas duas últimas programações, é mais difícil entender o fluxo de desvios do que o programa com apenas uma linha **IF...THEN...ELSE**.

DIVIRTA-SE COM LABIRINTOS

Jogos com labirintos complicados exigem programas muito longos. Mas você pode construir alguns bem simples com pouco mais do que um laço e declarações DATA.

Jogos com labirintos exerceram sempre um grande fascínio sobre usuários de computadores, e variações do tão conhecido "Come-come" (*PacMan*) continuam a aparecer constantemente no mercado.

Este artigo lhe ensina a entrar na onda e fazer um emocionante jogo de labirinto.

O labirinto, nesta primeira tentativa, não inclui "inimigos" ou obstáculos, que exigiriam um programa bem maior. Mas ele mostra como fazer para que os personagens do seu jogo não atravessem as paredes e inclui também cronometragem, manutenção de um placar e uma rotina de registro de recordes, para dar mais competitividade ao jogo.

S

A maneira mais fácil de compreender como os jogos com labirinto funcionam nos micros da linha Sinclair Spectrum, como o TK90X, é ir por etapas:

```
100 FOR n=3 TO 17
110 READ a$
120 FOR m=7 TO 21
130 PRINT AT n,m;"."
140 IF a$(m-6)="p" THEN  PRINT
 PAPER 1; INK 1;AT n,m;" "
150 NEXT m
160 NEXT n
9000 DATA "ppppppppppppppp"
9010 DATA "p.............p"
9020 DATA "p.pp.pp.pp.pp.p"
9030 DATA "p.p.........p.p"
9040 DATA "p...p.p.p.p...p"
9050 DATA "p.ppp.p.p.ppp.p"
9060 DATA "p.....p.p.....p"
9070 DATA "pppp.pp.pp.pppp"
9080 DATA "p.....p.p.....p"
9090 DATA "p.ppp.p.p.ppp.p"
9100 DATA "p...p.p.p.p...p"
9110 DATA "p.p.........p.p"
9120 DATA "p.pp.pp.pp.pp.p"
9130 DATA "p.............p"
9140 DATA "ppppppppppppppp"
```

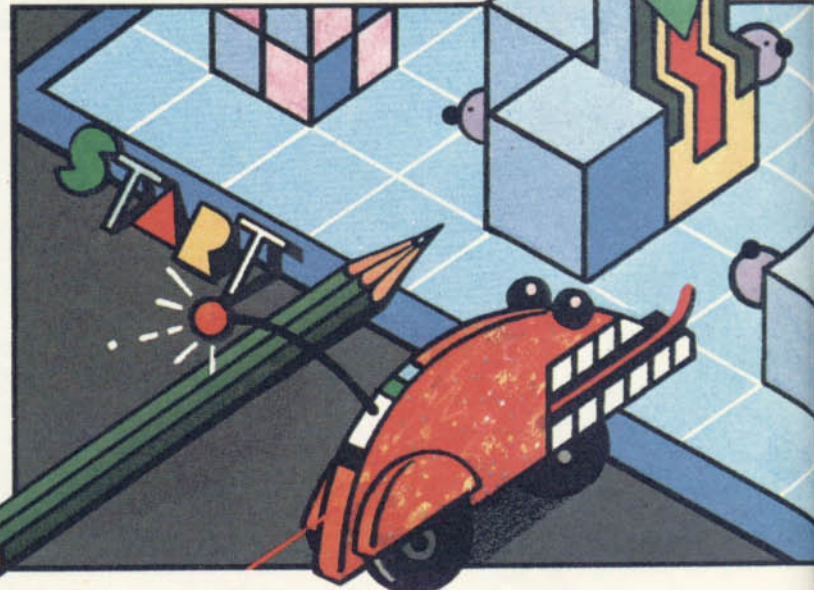
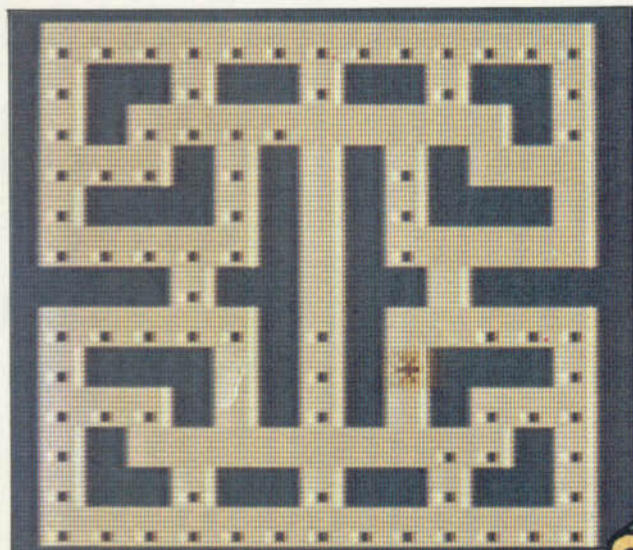
As linhas 100, 120, 150 e 160, usando um par dos nossos já conhecidos laços **FOR...NEXT**, determinam os limites do labirinto. A linha 130 imprime um ponto em cada posição da tela dentro desses limites.

As linhas 110 e 140 lêem os dados das linhas 9000 a 9140 e substituem o ponto por um espaço em fundo azul, onde o padrão dos dados mostrar um "p".

Isto é possível porque o TK90X nunca coloca dois caracteres na mesma posição; então, o último toma o lugar do primeiro.

O "COME-COME"

Um labirinto é inútil se não há ninguém ou algo para andar dentro dele. Então execute o programa acima para ver como fica e depois acione as seguintes linhas:

```
  50 LET x=10
  60 LET y=14
1000 PRINT  PAPER 6; INK 2;AT x
,y;"*"
1010 LET xx=x
1020 LET yy=y
1030 IF INKEY$="" THEN  GOTO 10
30
1040 IF INKEY$="w" AND ATTR (x-
1,y)<>9 THEN  LET x=x-1
1050 IF INKEY$="z" AND ATTR (x+
1,y)<>9 THEN  LET x=x+1
1060 IF INKEY$="a" AND ATTR (x,
```

Labirinto no TK-90X: com um programa bem curto.

```
y-1)<>9 THEN  LET y=y-1
1070 IF INKEY$="s" AND ATTR (x,
y+1)<>9 THEN  LET y=y+1
1080 PRINT  INK 7;AT xx,yy;" "
1090 GOTO 1000
```

Como todos os personagens famosos dos jogos, nosso amigo asterisco é posicionado em x,y e se movimenta, aos seus comandos, por uma série de linhas usando **INKEY$**.

Aqui, no entanto, há uma determinada diferença vital: ele só pode mover-se para uma posição se esta não for azul. Ou seja, se **ATTR** não for igual a 9, sendo 9 o valor numérico da cor com a qual você imprimiu as linhas do seu labirinto.

No caso de você querer criar um labirinto de outra cor, veja como chegar ao valor de **ATTR**:

1. Tome o valor da cor do caractere (**INK**), como está no teclado do computador — neste caso 1.

2. Tome o valor da cor da tela (**PAPER**) — neste caso 1 — e multiplique por 8. Resultado, 8.

3. Adicione 64 se a área em questão é clara (**BRIGHT**). Neste caso, resultado 0.

4. Adicione 128 se a área está piscando (**FLASH**). No caso, resultado 0.

Some então todos os números. O resultado será o valor de **ATTR**.

A função **ATTR** no TK90X tem, entretanto, outros usos além de impedir que o seu personagem atravesse paredes sólidas.

Em um labirinto maior, você poderia usar a mesma idéia para fazer com que ele exploda ao entrar em uma zona proibida. Para um labirinto que use caracteres (**INK**) vermelhos sobre fundo (**PAPER**) vermelho, você necessitaria uma linha começando com:

```
IF ATTR=18 THEN...
```

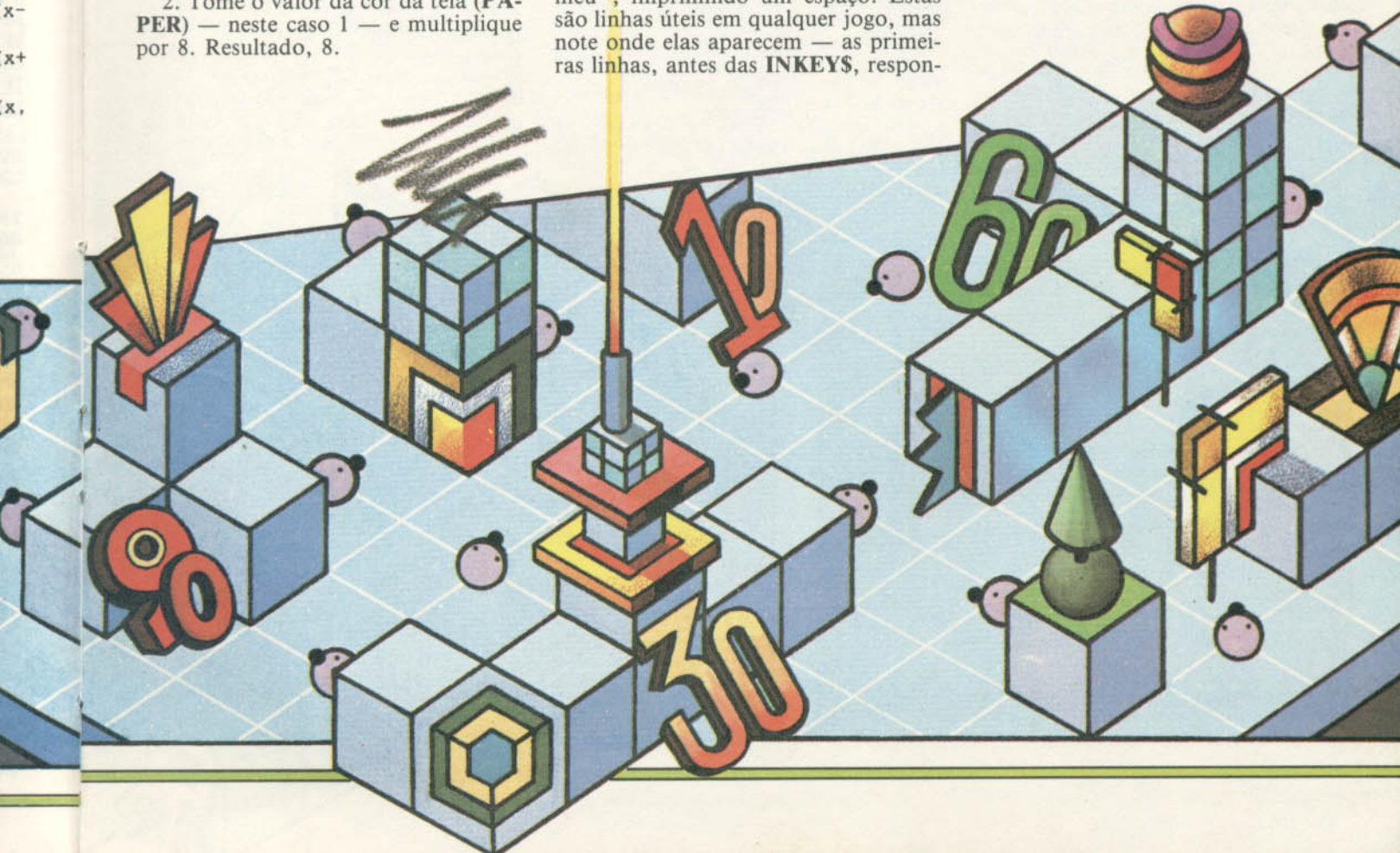
As linhas 1010, 1020 e 1080 definem a posição que o personagem acabou de deixar e apagam o ponto que ele "comeu", imprimindo um espaço. Estas são linhas úteis em qualquer jogo, mas note onde elas aparecem — as primeiras linhas, antes das **INKEY$**, responsáveis pela ação, e a linha final, logo após estas.

A linha 1090 é temporária e será reescrita depois. Seu propósito é permitir que você rode o programa e o teste. Sem fechar o laço, o caractere se moveria apenas uma vez.

TEMPO E PLACAR

Para que esse jogo possa ser jogado, sem contar com "inimigos" que exigiriam um programa muito complexo, o melhor é uma rotina de cronometragem e placar.

Adicione estas linhas:

```
  10 LET bt=100000
  40 LET s=0
 990 POKE 23672,0: POKE 23673,0
1025  IF s=110 THEN  GOTO 2000
1090 IF ATTR (x,y)<>63 THEN  LE
T s=s+1: SOUND .005,10
```

```
2000 LET t=(PEEK 23672+256*PEEK
 23673)/50
2010 PRINT AT 1,6;t;" SEGUNDOS
"
2020 IF t<bt THEN  LET bt=t
2030 PRINT AT 19,4;"Melhor temp
o: ";bt;" segundos"
```

Acrescentando uma linha temporária. Lembre-se de apagá-la depois:

```
1100 GOTO 1000
```

O placar é muito simples. A linha 40 coloca a contagem inicial em zero. A linha 1090 adiciona 1 cada vez que o "Come-come" devora mais um ponto, ou melhor, sempre que ele cruza um espaço em que **INK** e **PAPER** são ambos brancos — **ATTR** novamente.

E a linha 1025, que entra em ação quando todos foram comidos, faz com que o computador verifique quanto tempo o jogador levou para chegar a esse ponto.

Já a cronometragem é um pouco mais complicada. Até que você compreenda os comandos **PEEK** e **POKE**, que serão explicados num artigo próximo, procure assimilar o procedimento que se segue.

Em resumo, a linha 990 zera o relógio do TK90X ao jogar o valor 0 num endereço de memória apropriado. A linha 2000 conta o número de linhas formadas pela tela desde que o jogo começou e então divide por 50 — o número de linhas por segundo em um televisor.

Ao mesmo tempo, a linha 10 determina o "melhor tempo" inicial em 100000, muito mais que qualquer um levaria, de forma que ele será batido. A linha 2020 compara o novo tempo com o "melhor tempo".

OUTRA VEZ?

Para dar ao jogador a chance de outra tentativa, use estas novas linhas. Primeiro, pressione **<CAPS SHIFT>** e **<BREAK>** e, então, **<ENTER>**:

```
2040 PRINT AT 20,2;"Pressione q
ualquer tecla para   jogar de n
ovo.";
2050 IF INKEY$<>"" THEN  GOTO 2
050
2060 IF INKEY$="" THEN  GOTO 20
60
2070 RESTORE
2080 GOTO 40
```

OUTROS LABIRINTOS

Se você quiser tentar outros labirintos do mesmo tamanho, pode fazê-lo, mudando o padrão de letras "p" das linhas **DATA**, da linha 9000 em diante. Se fizer isso, assegure-se de ter dados suficientes para completar o labirinto ou terá uma mensagem de erro antes que apareça na tela.

T

Estas primeiras linhas formam o labirinto propriamente dito:

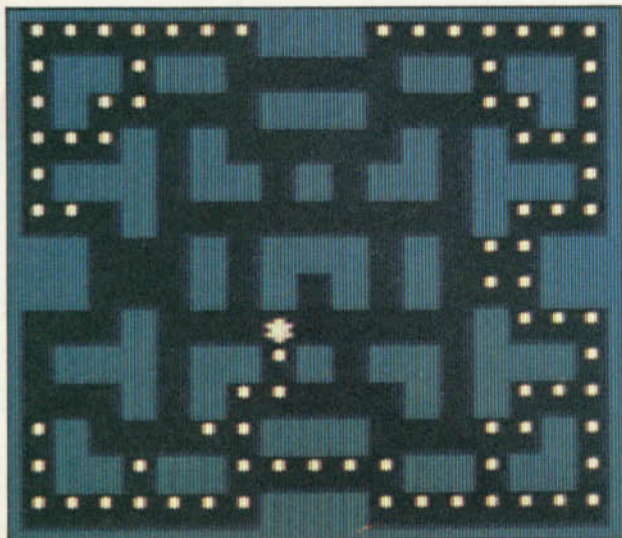
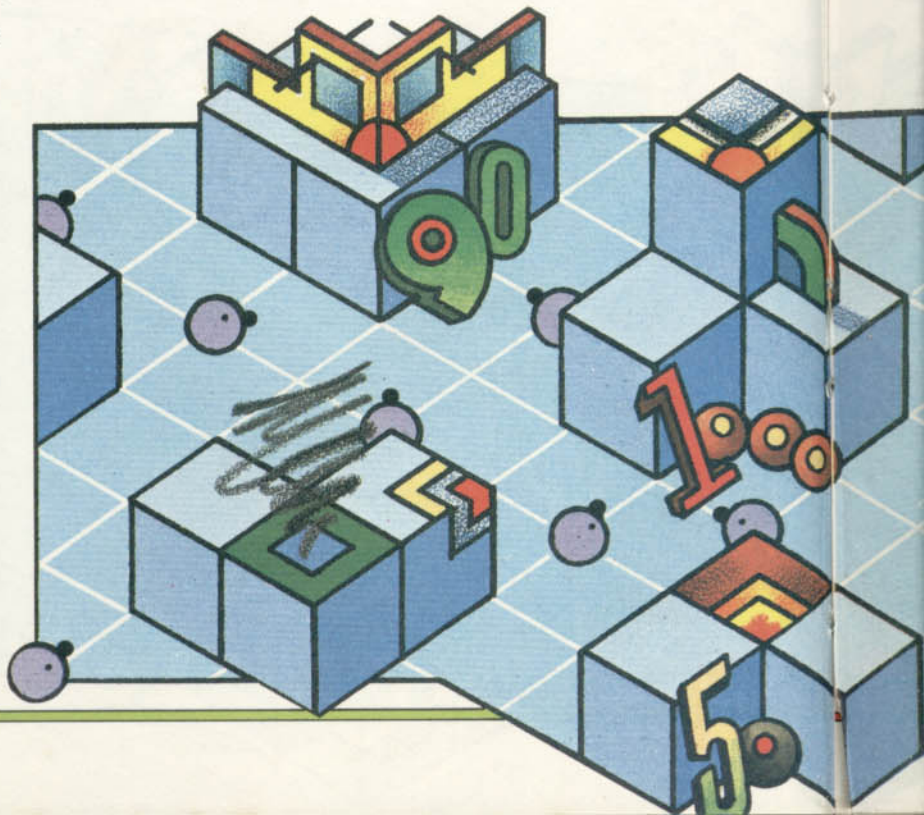
```
30 CLS4
40 LET P=297
1000 FOR N=0 TO 15
1010 READ A$
1020 PRINT @(N*32)+6,A$;
1030 FOR M=0 TO 18
1040 IF PEEK(1024+(N*32)+6+M)=6
5 THEN POKE(1024+(N*32)+6+M),17
5
1050 NEXT M
1060 NEXT N
2000 DATA AAAAAAAAAAAAAAAAAAA
2010 DATA A.......AAA.......A
2020 DATA A.AA.AA.....AA.AA.A
2030 DATA A.A.....AAA.....A.A
2040 DATA A...A.A.....A.A...A
2050 DATA A.AAA.AA.A.AA.AAA.A
2060 DATA A...A.........A...A
2070 DATA AAA...A.AAA.A...AAA
2080 DATA AAA...A.AAA.A...AAA
2090 DATA A...A.........A...A
2100 DATA A.AAA.AA.A.AA.AAA.A
2110 DATA A...A.A.....A.A...A
2120 DATA A.A.....AAA.....A.A
2130 DATA A.AA.AA.....AA.AA.A
2140 DATA A.......AAA.......A
2150 DATA AAAAAAAAAAAAAAAAAAA
```

A forma do labirinto está contida nas linhas de dados (**DATA**), de 2000 a 2150. Nesse estágio os limites dele são representados por letras "A".

Antes que os dados sejam lidos, a linha 30 limpa a tela e a coloca na cor vermelha, cujo código é 4. A linha 40, por sua vez, coloca o asterisco na posição inicial.

As linhas de 1000 a 1060 desenham o labirinto na tela. Ao mesmo tempo, sempre que o laço **FOR...NEXT** das linhas 1000 e 1060 é executado, o programa lê a próxima linha de dados (**DATA**).

A linha 1010 lê os dados e os chama de A$. A linha 1020 coloca-os na tela.

A linha 1040 verifica a porção da memória da máquina que corresponde a cada posição de tela. Se o endereço de memória contém o número 65, temos um "A" na tela, e ele é mudado para um bloco azul (código 175). As linhas 1030 e 1050 fornecem o *loop* com que

A versão para o TRS-Color mostra o tempo empregado.

a linha 1040 verifica todas as posições da tela.

Agora adicione essas linhas e você poderá deslocar um homenzinho pelo labirinto.

```
1080 PRINT @P,"*";
1090 LET LP=P
1100 IF PEEK(340)=247 AND PEEK(
1023+P)<>175 THEN LET P=P-1
1110 IF PEEK(338)=247 AND PEEK(
1025+P)<>175 THEN LET P=P+1
1120 IF PEEK(338)=251 AND PEEK(
992+P)<>175 THEN LET P=P-32
1130 IF PEEK(342)=253 AND PEEK(
1056+P)<>175 THEN LET P=P+32
1150 PRINT @LP," ";
1170 GOTO 1080
```

Esta parte do programa permite que você mova o seu caractere pelo labirinto. Ela tem algumas particularidades. O homem só pode ser movimentado se o ponto para onde se quer levá-lo não contém um bloco azul. A linha 1100 verifica o conteúdo de duas posições de memória: a que corresponde à tecla "Z" e a que corresponde à posição de tela em que o homem vai entrar. Se a tecla "Z" (código 223) está sendo pressionada e o endereço de tela não contém um bloco azul (código 175), então ele pode se movimentar.

As teclas para controle de movimento são as mesmas já utilizadas: "Z" para a esquerda, "X" para a direita, "P" para cima e "L" para baixo. Tente andar com o asterisco pelo labirinto de forma que os pontos sejam apagados. Acrescente estas linhas e marque o tempo necessário para "comer" os pontos:

```
20 LET FA=999999
50 RESTORE
60 LET D=0
1070 TIMER=0
1140 IF PEEK(1024+P)=110 THEN P
LAY "T80B":LET D=D+1
1160 IF D=153 THEN GOTO 1180
1180 PRINT @P," ";
1190 TI=TIMER/50
1200 PRINT @52,"TEMPO=";
1210 PRINT @84,TI;"SEG ";
1220 IF FA>TI THEN FA=TI
1230 PRINT @212,"MELHOR TEMPO";
1240 PRINT @244,FA;"SEG ";
1250 PRINT @340,"PRESSIONE";
1260 PRINT @372,"<ENTER>";
1270 PRINT @404,"PARA COMECAR";
1280 PRINT @436,"DE NOVO";
1290 LET IN$=INKEY$:IF IN$="" T
HEN GOTO 1290
1300 IF IN$=CHR$(13) THEN GOTO
40
1310 GOTO 1290
```

Quando você executa este programa pode verificar à direita do labirinto o tempo gasto para remover os pontos e também o melhor tempo.

Inicialmente, o melhor tempo é colocado em 999999 segundos pela linha 20. A linha 60 zera o placar, e a linha 1070 zera o "cronômetro".

A linha 1140 examina a posição em que o asterisco está, para ver se há um ponto. Em caso positivo, o comando **PLAY** é usado para emitir um "*blip*" enquanto o ponto é apagado. Ela também adiciona um ponto a D, que contém o número de pontos "comidos".

A linha 1160 verifica se não há mais pontos na tela. Se estes terminaram, o programa pula para a linha 1180, que apaga o asterisco.

A declaração **TI = TIMER/50** na linha 1190 pára o relógio e divide o que está sendo lido por 50, para transformá-lo em segundos.

As linhas 1200 e 1210 mostram o tempo na tela.

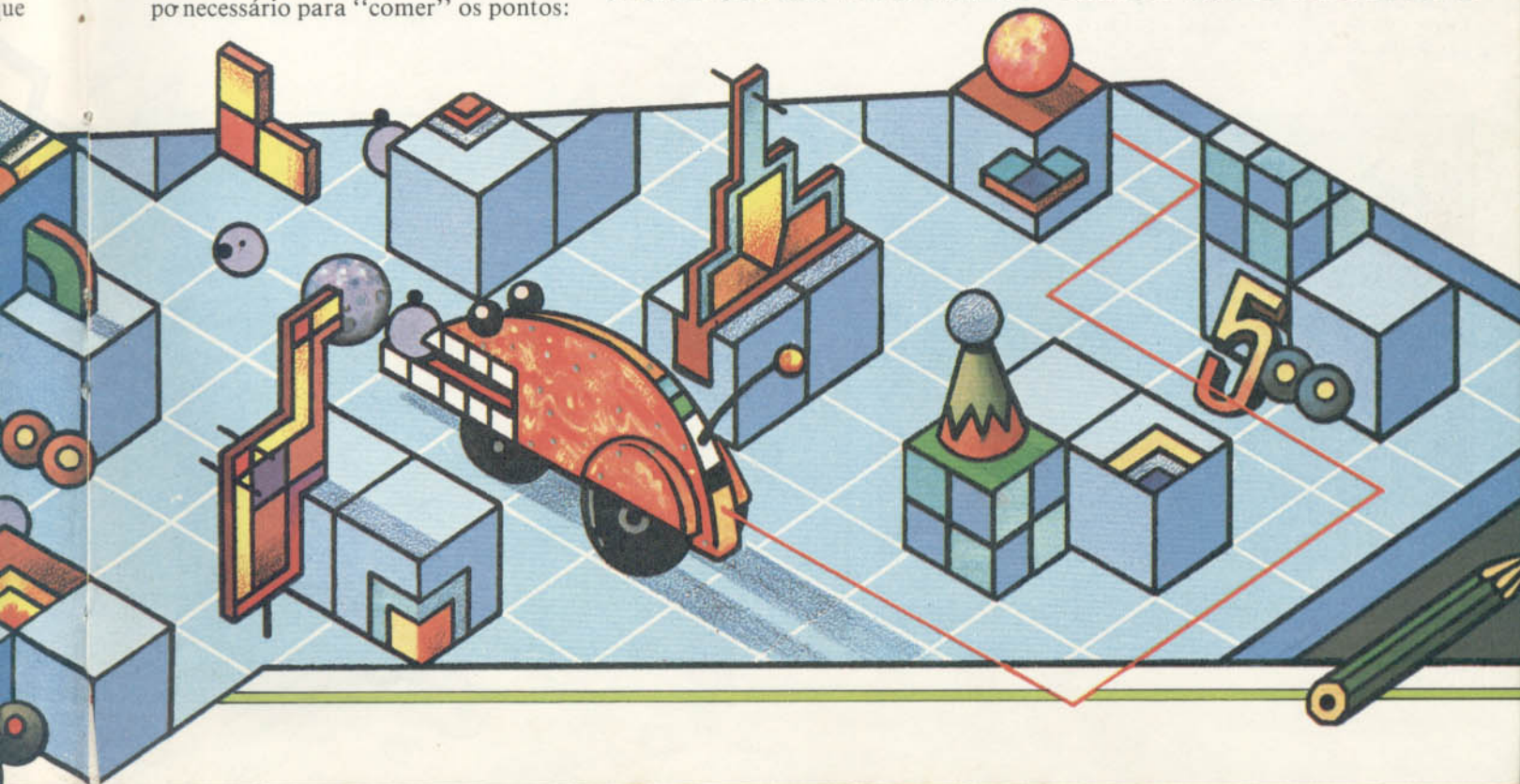
A linha 1220 checa se o melhor tempo (FA) é maior que o obtido na última contagem. Se for, ele é igualado a este. As linhas 1230 e 1240 mostram o melhor tempo, antes que uma mensagem oriente o jogador a teclar <**ENTER**> para recomeçar.

A linha 1290 aguarda, até que uma tecla seja pressionada, e a 1300 verifica se foi <**ENTER**>. Neste caso o jogo recomeça. O **RESTORE** na linha 50 permite que os dados (**DATA**) possam ser lidos novamente. Se <**ENTER**> não foi teclada, o computador continua esperando outra entrada.

A tecla <**BREAK**>, finalmente, termina o jogo.

CONSTRUA SEUS PROPRIOS LABIRINTOS

Use papel quadriculado para desenhar seus labirintos. Tenha sempre em mente o tamanho da tela do computador: 32 por 16 caracteres.

Desenhado o novo labirinto, altere o conteúdo das linhas **DATA**. Se o novo labirinto for maior ou menor que o antigo, você terá que modificar o tamanho do laço **FOR...NEXT** (linhas 1000 e 1030). Conte quantas linhas o novo labirinto ocupa e subtraia 1. Coloque o resultado no lugar do último número da

linha 1000. Da mesma forma, faça a contagem do número de posições da primeira linha e subtraia 1. Coloque, em seguida, esse número no lugar de 18 no fim da linha 1030.

Finalmente, conte quantos pontos existem no seu novo labirinto. Se encontrar um número diferente de 153, altere a linha 1160 também, substituindo 153 pelo novo número.

MSX

Estas primeiras linhas constituem o labirinto propriamente dito:

```
10 SCREEN 1:COLOR 1,3,3
40 LET P=247
70 VPOKE BASE(6)+5,26
80 VPOKE BASE(6)+1,102
90 LOCATE 0,3
100 FOR N=0 TO 15
110 READ A$
120 PRINT TAB(4);A$
130 FOR M=0 TO 18
140 IF VPEEK(BASE(5)+102+N*32+M
)=65 THEN VPOKE BASE(5)+102+N*3
2+M,8
150 NEXT M
160 NEXT N
1000 DATA AAAAAAAAAAAAAAAAAAA
1010 DATA A.......AAA.......A
1020 DATA A.AA.AA.....AA.AA.A
1030 DATA A.A.....AAA.....A.A
1040 DATA A...A.A.....A.A...A
1050 DATA A.AAA.AA.A.AA.AAA.A
1060 DATA A...A.........A...A
1070 DATA AAA...A.AAA.A...AAA
1080 DATA AAA...A.AAA.A...AAA
1090 DATA A...A.........A...A
1100 DATA A.AAA.AA.A.AA.AAA.A
1110 DATA A...A.A.....A.A...A
1120 DATA A.A.....AAA.....A.A
1130 DATA A.AA.AA.....AA.AA.A
1140 DATA A.......AAA.......A
1150 DATA AAAAAAAAAAAAAAAAAAA
```

As linhas de dados (**DATA**), de 1000 a 1150 encerram a forma do labirinto. Os limites deste são representados por letras "A".

Antes que os dados sejam lidos, a linha 10 limpa a tela, introduzindo-a no modo gráfico de 32 colunas (comando **SCREEN**), e fixa a cor de fundo e de frente (comando **COLOR**). A linha 40 define a posição inicial do "Come-come" (que será representado por um asterisco).

As linhas de 100 a 160 desenham o labirinto na tela. Sempre que o laço **FOR...NEXT** das linhas 100 e 160 é executado, o programa lê a próxima linha de dados (**DATA**). A linha 110 lê os dados, denominando-os de A$. A linha 120 coloca-os na tela.

Todos esses pontos e letras "A" combinados não formam, contudo, um labirinto de verdade. Assim, a linha 140 verifica a porção da memória da máquina que corresponde a cada posição de tela. Isso é feito com a função **VPOKE BASE (5)**, que torna acessível a memória dedicada de vídeo do MSX, na chamada página 5. Se o endereço de memória contém o número 65, isso quer dizer que temos um "A" na tela e ele é mudado para um bloco azul (código 8). As linhas 130 e 150 fornecem o laço que permite que a linha 140 verifique todas as posições da tela.

Agora adicione estas linhas e desloque um homenzinho pelo labirinto.

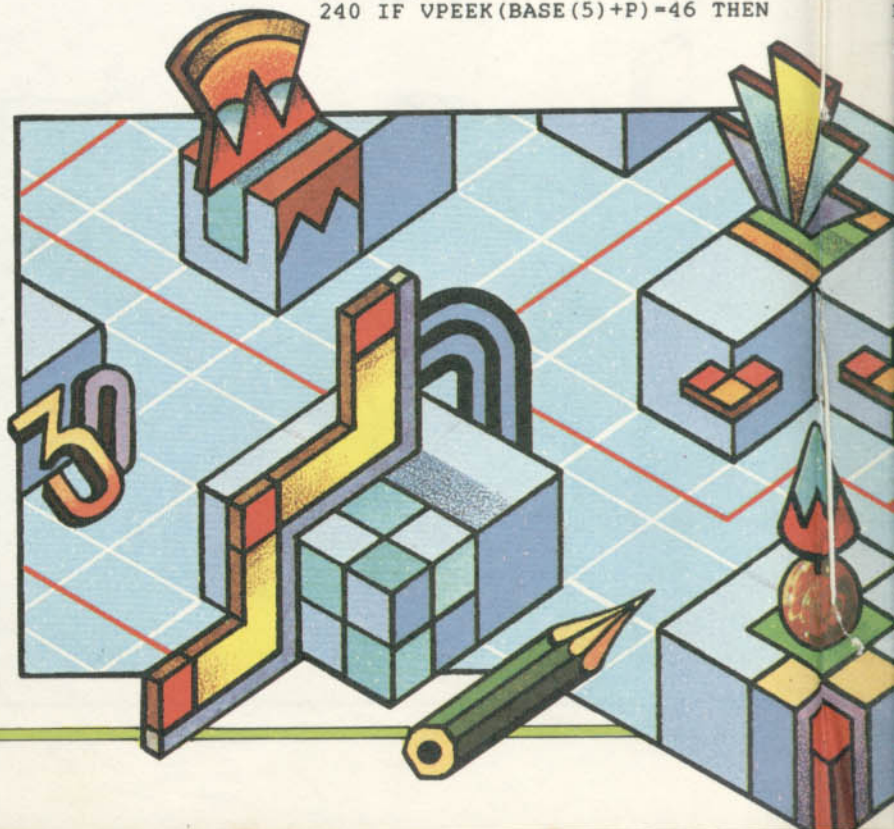
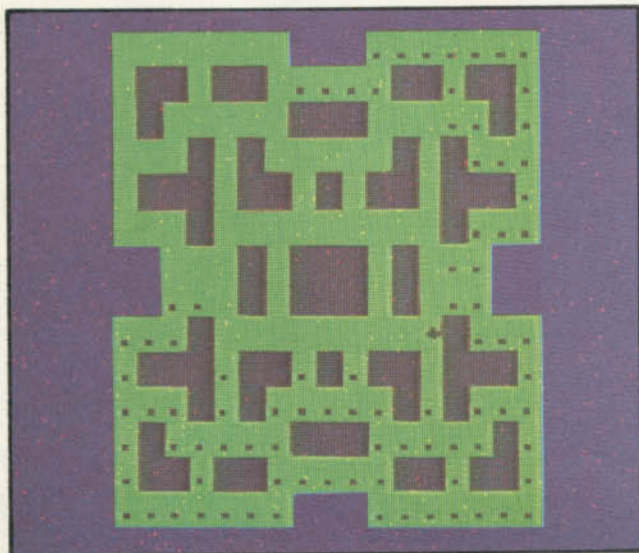
```
60 VPOKE BASE(6)+31,170
180 VPOKE BASE(5)+P,ASC("*")
190 LET LP=P
195 K$=INKEY$:IF K$="" THEN 195
200 IF K$=CHR$(29) AND VPEEK(BA
SE(5)+P-1)<>8 THEN LET P=P-1
210 IF K$=CHR$(28) AND VPEEK(BA
SE(5)+P+1)<>8 THEN LET P=P+1
220 IF K$=CHR$(30) AND VPEEK(BA
SE(5)+P-32)<>8 THEN LET P=P-32
230 IF K$=CHR$(31) AND VPEEK(BA
SE(5)+P+32)<>8 THEN LET P=P+32
250 VPOKE BASE(5)+LP,255
270 GOTO 180
```

O movimento do caractere pelo labirinto só é possível se o ponto para onde se quer levá-lo não contém um bloco azul. A linha 200 verifica se uma tecla foi apertada e qual é o conteúdo de duas posições de memória: a que corresponde à tecla de movimentação do cursor "flecha para baixo" e a que corresponde à posição de tela em que o homem vai entrar. Se a tecla (código 29) está sendo pressionada, e o endereço de tela não contém um bloco azul (código 8), ele pode se movimentar.

As teclas para controle de movimento são as usadas para a movimentação do cursor (marcadas com as flechas nas quatro direções). Experimente deslocar o asterisco e "comer" (apagar) os pontinhos.

Adicione estas linhas e marque o tempo para "comer" os pontos:

```
20 FA=999999!
30 RESTORE
50 LET D=0
170 TIME=0
240 IF VPEEK(BASE(5)+P)=46 THEN
```

Um labirinto bem fácil para o MSX.

```
SOUND 7,7:SOUND 6,0:SOUND 8,15
:SOUND 8,0:LET D=D+1
260 IF D=153 THEN GOTO 280
280 VPOKE BASE(5)+P,255
290 LET TI=INT(TIME/50)
300 LOCATE 7,1:PRINT "TEMPO=";T
I;"SEG  ";
310 IF FA>TI THEN FA=TI
320 LOCATE 3,20:PRINT "MELHOR T
EMPO=";FA;"SEG  ";
330 LOCATE 7,21:PRINT "Aperte R
ETURN ";
390 LET K$=INKEY$:IF K$="" THEN
 390
400 IF K$=CHR$(13) THEN GOTO 30
410 GOTO 390
```

O tempo gasto para remover os pontos, assim como o melhor tempo, é mostrado à direita do labirinto, quando você executa este programa. De início, o melhor tempo é colocado em 999999 segundos pela linha 20. A linha 50 zera o placar, e a linha 170 zera o "cronômetro".

A linha 240 examina a posição em que o asterisco está, para ver se há um ponto. Em caso positivo, o comando **SOUND** é usado para emitir um "*blip*" enquanto o ponto é apagado. Ela também acrescenta um ponto a D, que contém o número de pontos "comidos". A linha 260 verifica se não há mais pontos na tela. Se estes terminaram, o programa pula para a linha 280, que apaga o asterisco. Acionamos aqui o comando **VPOKE**, como acima.

A declaração **TI=INT(TIME/50)**, agora na linha 290, pára o relógio e divide o que está sendo lido por 50 para transformá-lo em segundos. A linha 300 mostra o tempo na tela.

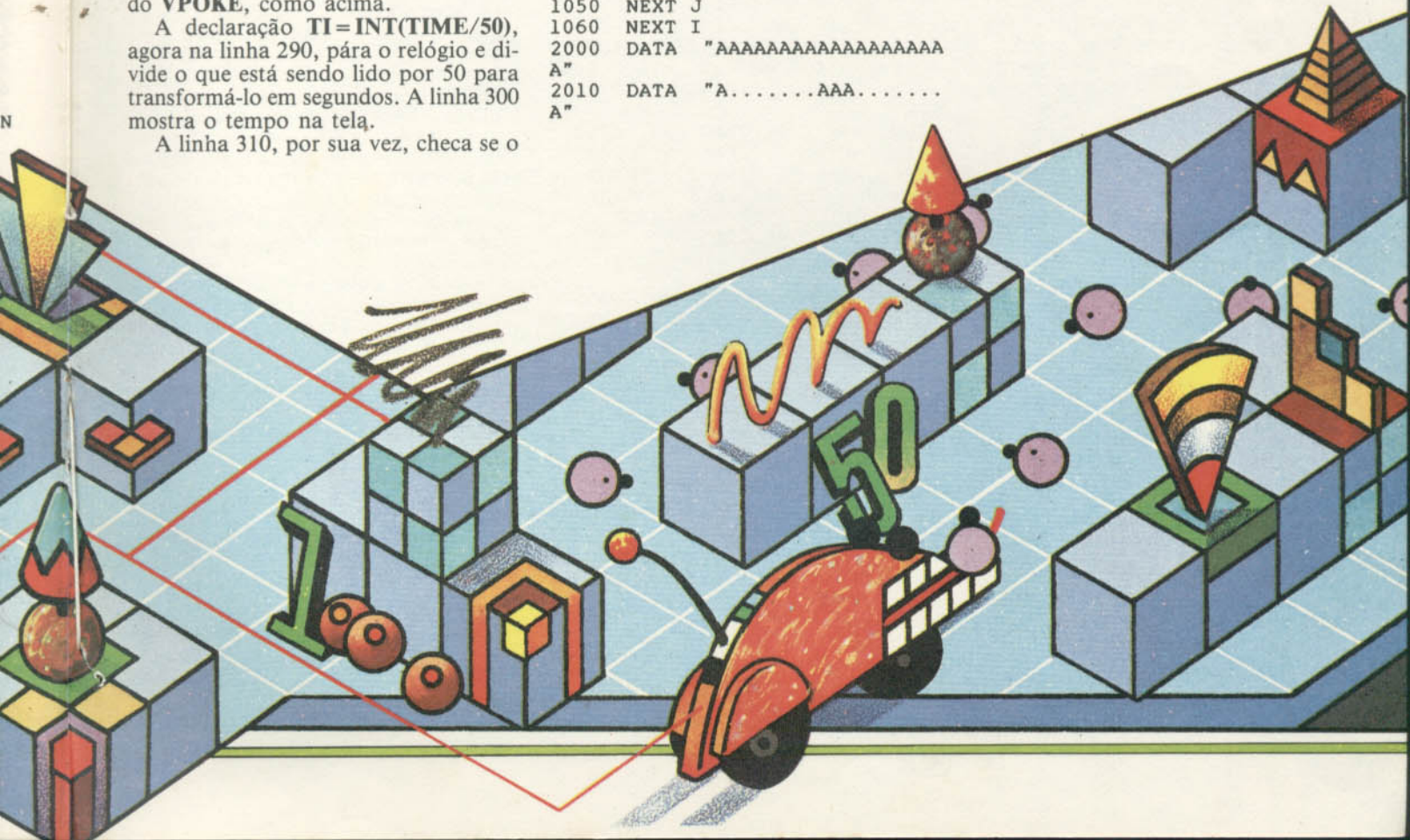
A linha 310, por sua vez, checa se o melhor tempo (FA) é maior que o tempo obtido na última contagem. Se foi, ele é igualado a este último. A linha 320 mostra o melhor tempo, antes que uma mensagem diga ao jogador para recomeçar, teclando <ENTER>.

A linha 390 aguarda até que uma tecla tenha sido pressionada, e a linha 400 verifica se foi <ENTER>. Neste caso, o jogo recomeça. O **RESTORE** na linha 30 permite que os dados (**DATA**) possam ser lidos novamente. Se alguma tecla diferente de <ENTER> foi pressionada, o computador continua esperando outra entrada.

O comando <CTRL><STOP> encerra o jogo.

Estas primeiras linhas formam o labirinto propriamente dito:

```
30   HOME : GR
40   LET M = 18: LET N = 4
1000  FOR I = 0 TO 15
1010  READ A$
1020  FOR J = 1 TO 19
1025  IF  MID$ (A$,J,1) = "A" T
HEN  COLOR= 3: PLOT 2 * J,2 * I
: PLOT 2 * J + 1,2 * I: PLOT 2
* J,2 * I + 1: PLOT 2 * J + 1,2
 * I + 1
1030  IF  MID$ (A$,J,1) = "." T
HEN  COLOR= 12: PLOT 2 * J,2 *
I
1050  NEXT J
1060  NEXT I
2000  DATA  "AAAAAAAAAAAAAAAAAA
A"
2010  DATA  "A.......AAA.......
A"
2020  DATA  "A.AA.AA.....AA.AA.
A"
2030  DATA  "A.A.....AAA.....A.
A"
2040  DATA  "A...A.A.....A.A...
A"
2050  DATA  "A.AAA.AA.A.AA.AAA.
A"
2060  DATA  "A...A.........A...
A"
2070  DATA  "AAA...A.AAA.A...AA
A"
2080  DATA  "AAA...A.AAA.A...AA
A"
2090  DATA  "A...A.........A...
A"
2100  DATA  "A.AAA.AA.A.AA.AAA.
A"
2110  DATA  "A...A.A.....A.A...
A"
2120  DATA  "A.A.....AAA.....A.
A"
2130  DATA  "A.AA.AA.....AA.AA.
A"
2140  DATA  "A.......AAA.......
A"
2150  DATA  "AAAAAAAAAAAAAAAAAA
A"
```

Os limites do labirinto são representados, neste estágio, tal como em exemplos anteriores, por letras "A". Da mesma forma, seus contornos estão contidos nas linhas de dados (**DATA**), de 2000 a 2150. Portanto, o programa só

funcionará se forem adicionadas essas linhas. É relativamente fácil alterar a forma para um novo labirinto: basta modificar as linhas de 2000 a 2150.

Antes que os dados sejam lidos, a linha 30 limpa a tela e a coloca num modo gráfico de baixa resolução. A linha 40 coloca o asterisco na posição incial.

As linhas de 1000 a 1060 desenham o labirinto na tela. A cada vez que o laço **FOR...NEXT** das linhas 1000 e 1060 é executado, o programa lê a próxima linha de dados (**DATA**). A linha 1010 lê os dados, e os chama A$. A linha 1025 coloca-os na tela, na forma de quadradinhos coloridos (cor = 3 onde houver a letra A, e cor = 12 em outros pontos).

Agora adicione estas linhas e você poderá deslocar, mais uma vez, um homenzinho pelo labirinto.

```
1080  COLOR= 1: PLOT 2 * M,2 *
N: PLOT 2 * M + 1,2 * N: PLOT 2
 * M,2 * N + 1: PLOT 2 * M + 1,
2 * N + 1
1090  LET PM = M: LET PN = N
1100  IF  PEEK ( - 16384) = 218
 AND  SCRN( 2 * (M - 1),2 * N)
<  > 3 THEN  LET M = M - 1
1110  IF  PEEK ( - 16384) = 216
 AND  SCRN( 2 * (M + 1),2 * N)
<  > 3 THEN  LET M = M + 1
1120  IF  PEEK ( - 16384) = 208
 AND  SCRN( 2 * M,2 * (N - 1))
<  > 3 THEN  LET N = N - 1
1130  IF  PEEK ( - 16384) = 204
 AND  SCRN( 2 * M,2 * (N + 1))
<  > 3 THEN  LET N = N + 1
1150  COLOR= 0: PLOT 2 * PM,2 *
 PN: PLOT 2 * PM + 1,2 * PN: PL
OT 2 * PM,2 * PN + 1: PLOT 2 *
PM + 1,2 * PN + 1
1170  GOTO 1080
```

Esta parte do programa dá a você a possibilidade de movimentar o caractere pelo labirinto. Para que haja movimento, porém, o ponto para onde se quer levar o caractere (nosso homenzinho) não deve conter nenhum bloco de cor 12.

A linha 1100, por exemplo, verifica o conteúdo de duas posições de memória: a que corresponde à tecla "Z" e a que corresponde à posição de tela em que o homem vai entrar. Se a tecla "Z" (código 218) está sendo pressionada, e o endereço de tela não contém um bloco gráfico, então ele pode se movimentar.

As teclas para controle de movimento são as mesmas já utilizadas anteriormente: "Z" para a esquerda, "X" para a direita, "P" para cima e "L" para baixo.

Marque o tempo necessário para "comer" todos os pontos, acrescentando estas linhas:

```
20  LET FA = 99999
50  RESTORE
60  LET D = 0
1070  LET TI = 0
1140  IF  SCRN( 2 * M,2 * N) =
12 THEN  LET W =  PEEK ( - 1633
6): LET D = D + 1
1150  COLOR= 0: PLOT 2 * PM,2 *
 PN: PLOT 2 * PM + 1,2 * PN: PL
OT 2 * PM,2 * PN + 1: PLOT 2 *
PM + 1,2 * PN + 1
1160  IF D = 153 THEN  GOTO 118
0
1170  GOTO 1080
1180  COLOR= 0: PLOT 2 * M,2 *
N: PLOT 2 * M + 1,2 * N: PLOT 2
 * M,2 * N + 1: PLOT 2 * M + 1,
2 * N + 1
1200  HTAB 9: VTAB 21: PRINT "S
eu tempo foi de ";TI / 5;" seg"
1210  IF FA > TI THEN  LET FA =
 TI
1220  HTAB 11: VTAB 22: PRINT "
Melhor tempo: ";FA / 5;" seg"
1230  HTAB 6: VTAB 23: PRINT "P
ressione RETURN para continuar"
1240  GET T$: IF  ASC (T$) <  >
 13 THEN  GOTO 1210
1250  GOTO 40
```

De início, o melhor tempo é colocado em 999999 segundos pela linha 20. A linha 60 zera o placar e a linha 1070 zera o "cronômetro". A linha 1095 aumenta uma posição no cronômetro. A linha 1140 examina a posição em que o asterisco está. Ela também adiciona um ponto a D, que contém o número de pontos que foram "comidos".

A linha 1160 verifica se não há mais pontos na tela (em número de 153). Se estes terminaram, o programa pula para a linha 1180, que apaga o asterisco.

A linha 1200 mostra o tempo na tela (que é convertido para segundos dividindo-se FA por 50).

A linha 1210 checa se o melhor tempo (FA) é maior que o tempo obtido na última contagem. Em caso positivo, ele é igualado a este último. A linha 1220 mostra o melhor tempo, antes que uma mensagem diga ao jogador para teclar **<ENTER>** para recomeçar (na linha 1230).

A linha 1290 aguarda até que uma tecla tenha sido pressionada, e a linha 1300 verifica se foi **<ENTER>**. Neste caso, o jogo recomeça. O **RESTORE** na linha 50 permite que os dados (**DATA**) possam ser lidos novamente. Caso tenha sido acionada uma tecla que não **<ENTER>**, o computador continua à espera de outra entrada. A tecla **<CTRL-C>** termina o jogo.

CONVERSÃO DO COLOR PARA O TRS-80

Os micros TRS-80 e TRS-Color têm interpretadores BASIC similares, no que diz respeito ao núcleo básico de comandos, funções e declarações. Por isso, é fácil converter programas de um modelo para outro. As maiores diferenças residem nos comandos de alta resolução gráfica, cor e som. Os comandos **CLS** (limpa telas), **SET** e **POINT** (pontos gráficos de resolução média) e **PRINT@** são semelhantes nas duas linhas. As diferenças de sintaxe do TRS-80 em relação ao Color são: o **CLS**, o **SET** e o **POINT** não têm parâmetro indicativo de cor de fundo (tela de texto), e as locações de tela usadas com o **PRINT@** produzem resultados diferentes.

A tela do Color tem dezesseis linhas por 32 colunas, e a do TRS-80, dezesseis linhas por 64 colunas. Como as posições de tela usadas pelo **PRINT@** são numeradas seqüencialmente, um **PRINT@ 162**, por exemplo, escreverá na segunda coluna da sexta linha do TRS-Color, e na 32.ª posição da terceira linha, no TRS-80.

Há uma forma simples de converter comandos **PRINT@** do Color para o TRS-80: se **N** for a locução de tela do Color, sua correspondente no TRS-80 será

```
INT(N/32)*64+N-32
```

para a segunda linha em diante (a que começa na posição 32 do Color). Se N < 32, não há necessidade da fórmula acima: a locução de tela do TRS-80 será o próprio N.

COMO DESCOMPLICAR SAVEs E LOADs

- APRENDA A CONECTAR O GRAVADOR
- COMANDO DE GRAVAÇÃO
- VERIFIQUE SUAS FITAS
- COMANDO DE LEITURA

Qualquer criança é capaz de fazer funcionar um gravador. Quando se trata de conectá-lo ao computador, porém, as coisas podem se complicar. Saiba como sair dessa, começando por ajustar os controles.

Alguns problemas podem surgir quando se quer gravar ou carregar programas com um gravador cassete. As causas desses problemas nem sempre são identificáveis. Mas é possível estabelecer algumas rotinas que diminuam as margens de erro.

SAIBA CONECTAR CORRETAMENTE

Alguns micros empregam um gravador especial do tipo digital, que tem uma ligação única e direta com a máquina; essa característica elimina qualquer problema na instalação. Mas a maioria dos computadores pode ser ligada a um gravador cassete, freqüentemente utilizando mais que um tipo de cabo. Normalmente, a ligação é feita por meio de um plugue do tipo DIN ligado a três plugues de pino (*jaques*), mas pode-se encontrar também um conector DID em cada uma das pontas, com ou sem jaques em paralelo. Se você comprar um cabo ou usar um próprio para áudio, assegure-se de que ele é adequado: caso não seja, aparecerão problemas como interferências em cabos não blindados.

Se você usa o gravador com freqüência, é aconselhável deixar o cabo permanentemente ligado dos dois lados.

AJUSTE OS CONTROLES

O passo inicial e mais importante na adequação com o computador é ajustar os controles de volume, tonalidade e outras características do gravador. Se você tem um gravador especial para computadores, siga as instruções que o acompanham.

Se seu gravador é do tipo comum, comece desligando todos os controles especiais, como filtros e sistemas de redução de ruído (*Dolby*), etc. Ajuste o controle de tonalidade para o máximo de agudos (a tonalidade é controlada ou por um dial numerado, ou por um interruptor de duas posições grave/agudo). Deixe a tonalidade assim sempre que você usá-lo com o computador. Escolha um volume médio (entre 50 e 60% do máximo) e tente carregar (com os comandos **LOAD, CLOAD**, etc., dependendo do computador) um programa já gravado, como por exemplo o da fita de demonstração que acompanha sua máquina. Se o programa não carrega, aumente o volume pouco a pouco e repita a operação, até conseguir um bom resultado.

Se você alcançar o volume máximo ainda sem sucesso, volte o seletor até um ponto levemente abaixo daquele em que você começou e repita toda a operação, diminuindo o volume aos poucos.

Caso o programa não carregue em nenhum nível de volume, verifique as conexões novamente. Tente outro cabo ou outra fita, ou peça emprestado um gravador adequado ao seu tipo de computador. Se, depois disso, ele continuar sem funcionar, procure o seu revendedor.

Quando tiver encontrado um volume que permita o carregamento, observe a posição do controle, estabeleça os limites superior e inferior com os quais você consegue carregar o programa e marque o ponto médio para uso posterior. Você pode usar o mesmo volume para gravação, mesmo porque muitos gravadores têm um controle automático desse volume.

Para testar se o volume é adequado para gravação, limpe a memória do computador (digite **NEW** e pressione **<ENTER>** a seguir) e digite este pequeno programa.

```
10 REM TESTE DE GRAVACAO
20 REM
30 REM
40 REM
50 REM FIM DO TESTE
```

EXTERMINADOR DE PROBLEMAS

— Utilize um gravador cassete monoaural (isto é, que não seja estéreo), de marca confiável e, se possível, reserve-o para uso exclusivo com o computador. Ligue-o sempre à rede elétrica, ao invés de usar pilhas, para assegurar uma rotação uniforme do motor.

— Evite usar aparelhos estereofônicos, a não ser que todos os seus recursos especiais possam ser desligados.

— Use fitas magnéticas para áudio de boa qualidade ou especiais para dados. Uma fita de má qualidade não resistirá por muito tempo a várias regravações.

— Procure instruções específicas na tela; o gravador deve ficar com a tecla **PLAY** acionada até que a operação de carga esteja completada.

— Altere as características de tonalidade, volume e outros controles se a carga do programa não for bem-sucedida.

— Afaste o gravador da TV ou monitor se um programa que já carregou uma vez não funcionar. Tente também uma outra gravação do programa para ver se funciona. Em caso afirmativo, a primeira fita deve ter sido danificada.

— Se for necessário ajustar o volume com freqüência, de programa para programa, marque o volume adequado na etiqueta da fita.

— Guarde suas fitas em lugar seco, livre de poeira e longe de fontes eletromagnéticas e do calor.

Siga então a rotina para gravação no computador, acionando **SAVE** ou **CSAVE**. Ao carregá-lo (comando **LOAD** ou **CLOAD**) e listá-lo (**LIST**) você obterá o programa de volta.

O COMANDO DE GRAVAÇÃO

O comando de gravação de fita não é um comando padrão do BASIC, variando muito entre os computadores. A maneira correta de manipulá-lo é geralmente explicada no manual que acompanha o micro. Alguns passos iniciais, porém, são importantes.

Antes de gravar qualquer programa, coloque uma fita de boa qualidade no gravador, tendo o cuidado de rodá-la um pouco para que a guia de plástico transparente, existente no começo (*leader*), não fique em contato com o cabeçote.

A seguir, escolha um nome adequado para seu programa. Os dados são armazenados na forma de "arquivos" de um tipo ou outro, independentemente do método real de armazenamento. Mas, para gravações feitas em fita, nomes especiais são empregados com o comando **SAVE**, de modo a identificar na fita qual o programa gravado (isso é útil para achar automaticamente o programa, depois, através do comando **LOAD**).

O nome do arquivo ou programa pode ser qualquer combinação de caracteres ou símbolos contidos no tamanho máximo permitido para uma linha no seu computador. Na maioria das máquinas, esse tamanho é de dez caracteres no máximo. Outros computadores, como o TRS-80, permitem apenas uma letra ou símbolo de identificação do programa: quando o nome é maior, os comandos **CSAVE** e **CLOAD** reduzem-no à primeira letra.

```
SAVE "NOMEPROG01"
SAVE "NomeProg01"
CSAVE "R"
```

Dos computadores considerados aqui, apenas os compatíveis com Apple II não exigem aspas.

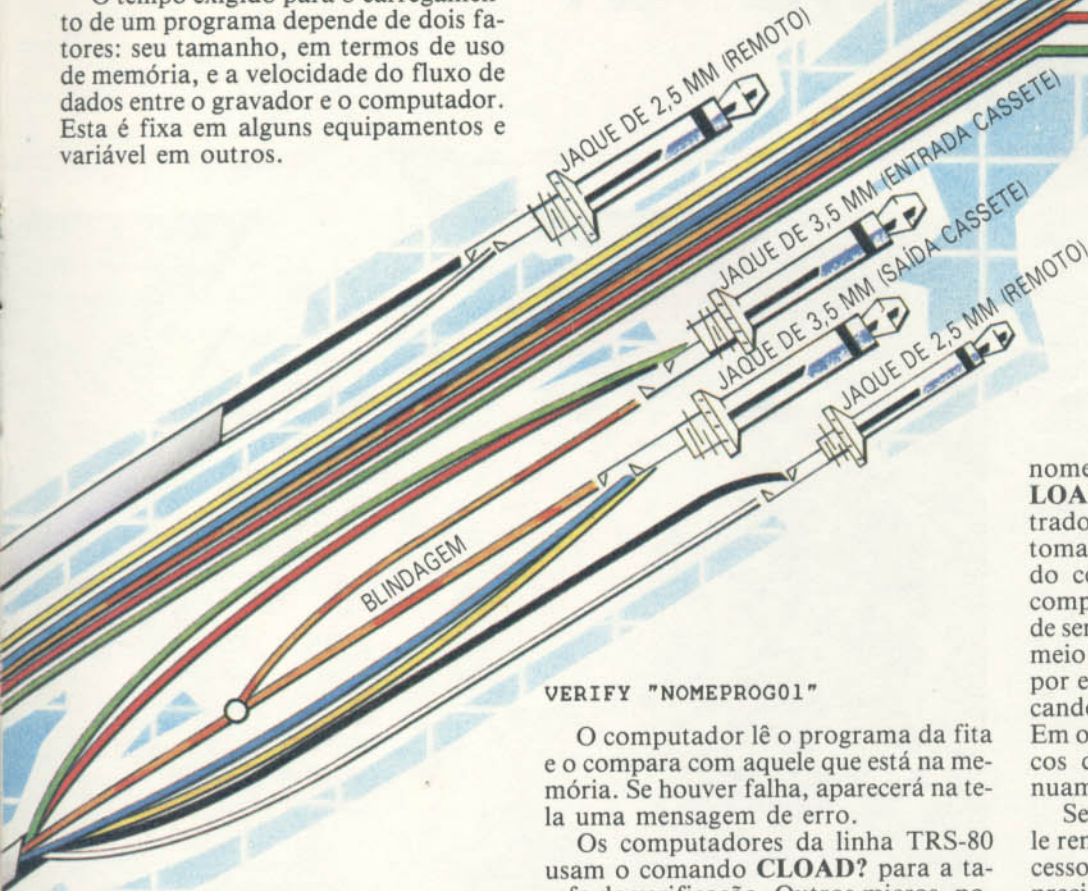
Os exemplos de comandos **SAVE** acima são válidos, mas um deles usa letras maiúsculas e minúsculas. Se você misturá-las num nome de programa, deve fazer o mesmo quando quiser carregar o programa.

```
SAVE "NOME.PROG"
SAVE "NOME/PROG"
SAVE "NOME(PROG)"
```

Se o seu computador está ligado ao controle remoto do gravador, ligue os controles para gravação, digite o nome do programa desejado após o comando **SAVE** e tecle **<ENTER>** ou **<RETURN>**. O computador assumirá o controle do gravador até que a operação de gravação esteja completada.

Se o controle é manual, digite o comando **SAVE** e o nome do programa, coloque o gravador em funcionamento e então tecle **<ENTER>** ou **<RETURN>**. Espere até que o sinal de

prontidão reapareça na tela e desligue o gravador.

O tempo exigido para o carregamento de um programa depende de dois fatores: seu tamanho, em termos de uso de memória, e a velocidade do fluxo de dados entre o gravador e o computador. Esta é fixa em alguns equipamentos e variável em outros.

Eis aqui dois tipos de cabo que você pode utilizar para conectar o seu computador a um gravador. Verifique em seu manual a disposição correta dos pinos (canto inferior esquerdo).

As gravações mais confiáveis são feitas na velocidade mais baixa. Altas velocidades usam menos fita e, obviamente, operam mais rápido. Nesse caso é importante usar fitas de alta qualidade.

VERIFIQUE AS GRAVAÇÕES

Uma maneira de verificar se um programa foi gravado adequadamente é carregá-lo e executá-lo. Quando a gravação é imperfeita o programa que está na memória (e que você quer gravar) é substituído por aquele que está na fita, e a gravação se perde. Se esta tiver sido bem-sucedida, não haverá problemas.

Muitos computadores contornam essa dificuldade fornecendo um comando verificador (por exemplo, **VERIFY**). O seu uso exige o rebobinamento da fita até o início do programa. Então executa-se o comando usando o mesmo nome que se deu ao programa ao gravá-lo. Por exemplo:

```
VERIFY "NOMEPROG01"
```

O computador lê o programa da fita e o compara com aquele que está na memória. Se houver falha, aparecerá na tela uma mensagem de erro.

Os computadores da linha TRS-80 usam o comando **CLOAD?** para a tarefa de verificação. Outros micros, porém (como os compatíveis com o Sinclair ZX-81), não têm qualquer função de verificação. Recorra, nesse caso, a programas de revistas especializadas ou vendidos à parte.

O COMANDO DE LEITURA

Se você começar com programas pré-gravados, seus primeiros problemas serão com o comando de leitura (**LOAD**). Podem existir várias formas do comando **LOAD**, mesmo para uma única máquina. Eis uma delas:

```
LOAD "NOMEPROG01"
```

Outros computadores utilizam a forma **CLOAD** (linha TRS-80), **LOADA** ou **LOADT** (TK-2000), etc.

O comando **LOAD** deve reproduzir o nome utilizado para gravação (**SAVE**) e verificação (**VERIFY**). Para que o computador leia os sinais da fita magnética, basta digitar **LOAD** e pressionar a tecla **PLAY** do gravador. Ele procurará pelo nome do programa e mostrará todos que encontrar pelo caminho, até encontrar aquele que confere com o nome indicado juntamente com o **LOAD**. Quando esse nome for encontrado, os dados que o seguem serão automaticamente carregados na memória do computador. Em muitos tipos de computador, o processo de leitura pode ser acompanhado na tela de vídeo por meio de indicadores. Na linha TRS-80, por exemplo, aparece um asterisco piscando no canto superior direito da tela. Em outras, o número de bytes ou de blocos de bytes lidos é mostrado continuamente.

Se o gravador está ligado ao controle remoto, a fita estaca ao final do processo de carregamento, mas você ainda precisa apertar a tecla **STOP** no gravador. Se o controle é manual, simplesmente aperte **STOP** quando o cursor reaparecer na tela. Evite que a tecla **PLAY** fique acionada desnecessariamente, porque isso pode provocar o desgaste prematuro ou deformação da polia tratora.

Quando você usar o comando **LOAD** ou equivalente, o programa que for carregado tomará o lugar de qualquer outro que esteja na RAM (Memória de Acesso Randômico) do computador. Por isso, assegure-se de ter gravado o programa resistente na memória antes de carregar o próximo.

Rebobine a fita até o início do programa-teste que foi apresentado lá em cima, e tente carregá-lo (**LOAD**) usando o nome que você escolheu. Em caso de dificuldade, consulte o guia de resolução de problemas, neste artigo.

Outras formas do comando **LOAD** são necessárias para tornar acessíveis dados em código de máquina (por exemplo, **BLOAD**), para realocar programas e para juntar um programa a outro (**MERGE**). Seu primeiro contato com elas pode ter sido em instruções de jogos comprados ao acaso.

APRENDA ARITIMÉTICA HEXADECIMAL

Os circuitos internos dos computadores utilizam, como vimos, apenas números binários para realizar operações aritméticas. Ora, esse sistema de numeração, composto dos dígitos 0 e 1, cria uma série de dificuldades de compreensão e manipulação para os operadores.

Em primeiro lugar, os números binários de oito e dezesseis bits (os mais usados em microcomputadores) tornam-se confusos e difíceis de reconhecer, com tantos 0s e 1s. Digitá-los no computador é uma empresa difícil, sujeita freqüentemente a erros e incorreções.

A maneira de contornar esses problemas consiste em fazer com que o operador ou programador em nível de máquina passe a utilizar um sistema numérico com outra base, mas que ainda seja próximo, em concepção, ao sistema binário utilizado pelo computador.

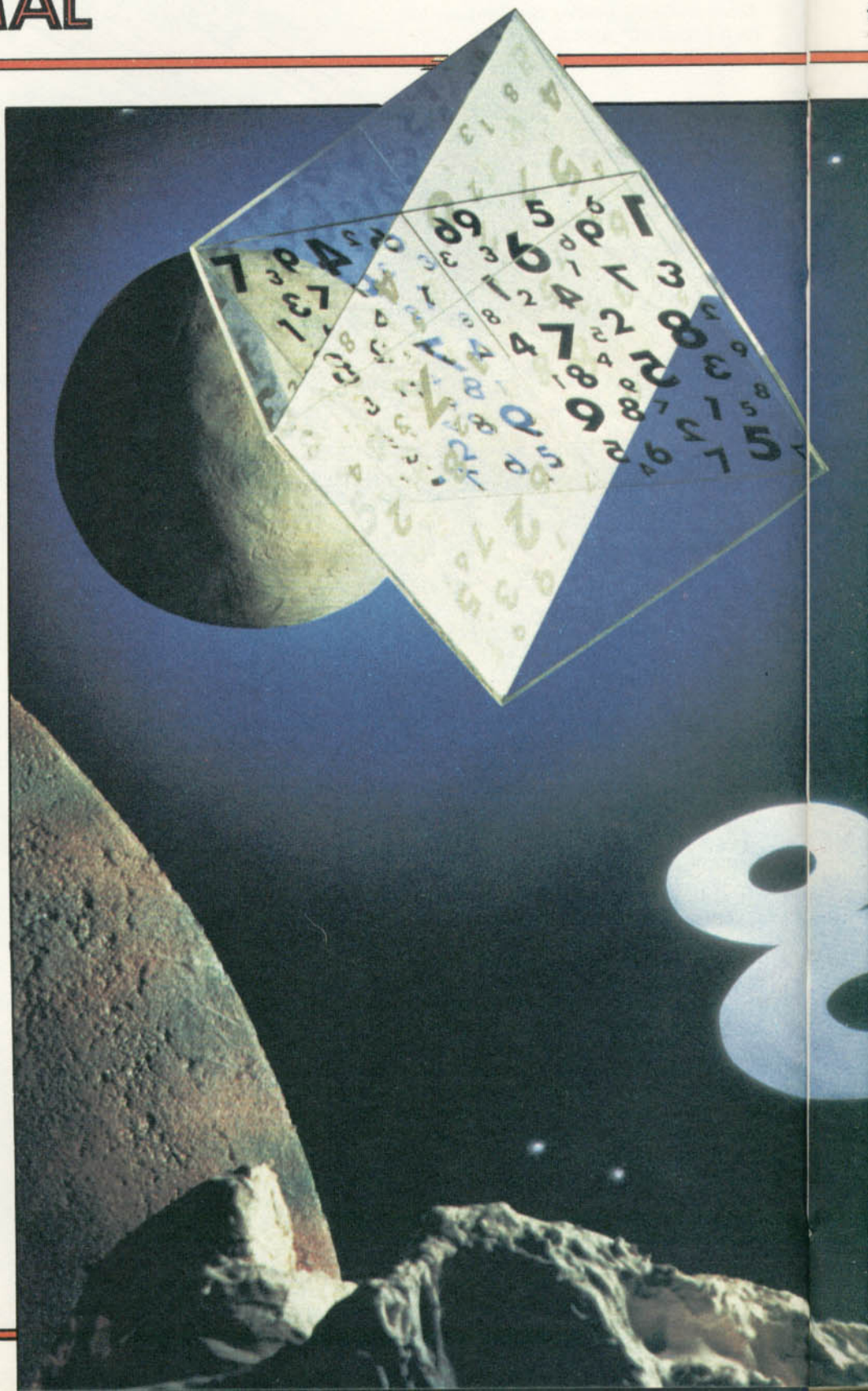
O sistema universalmente adotado para isso é o *hexadecimal* (ou hexa), no qual os números utilizam a base 16. E por que o hexa funciona tão bem com computadores?

Antes de mais nada, ele está suficientemente próximo do decimal para ser tolerado por nós, simples mortais. Mas, além disso, 16 é uma potência de 2 (como 8 também o é, em outro sistema muito usado com computadores, o *octal*). Isso significa que a conversão entre os sistemas — binários e hexa —, nos dois sentidos, é bem mais simples do que entre o primeiro e o sistema decimal.

A única dificuldade é que um sistema com base maior do que 10 precisa criar novos algarismos, além dos que existem entre 0 e 9. No sistema hexadecimal utilizam-se letras adicionais (sempre em maiúsculas). O 10 decimal é representado por A, o 11 por B, o 12 por C, o 13 por D, o 14 por E e o 15 por F.

Tudo se passa como se tivéssemos oito dedos em cada mão. Essa imagem pode parecer estranha ou mesmo monstruosa, pois dificilmente suportamos alterações em nossa aparência física. Na prática, porém, é isso que acontece.

COMO CONVERTER BINÁRIO EM HEXA

A conversão de números binários de oito bits (os mais utilizados em compu-

Se todos tivéssemos dezesseis dedos, o sistema numérico mais comum seria certamente o hexadecimal. Embora isso não aconteça, esse sistema é fundamental para o código de máquina.

- POR QUE O SISTEMA HEXADECIMAL É UTILIZADO
- CONTANDO DE 16 EM 16
- A RELAÇÃO ENTRE OS SISTEMAS BINÁRIO E HEXADECIMAL
- CONVERSÕES FÁCEIS DE BINÁRIO PARA HEXA
- CONVERSÃO DE DECIMAL PARA HEXA

tadores pessoais) para o sistema hexadecimal é bastante fácil. Primeiro, você divide o número binário em duas partes de quatro bits cada. Em seguida, cada uma dessas partes é transformada em um dígito hexadecimal. O número hexa final é composto por esses dois dígitos. Da mesma forma, converte-se um número binário de dezesseis bits hexa dividindo-o em quatro segmentos de quatro bits cada.

Já a conversão de decimal para hexa é mais complicada, embora não seja tão difícil que você não possa realizá-la. Para se fazer isso, você precisa dividir o número decimal por 16, sucessivamente. Os restos dessa divisão formarão os dígitos do número em hexa.

Quando se divide o decimal 1226 por 16, o resto será 10. O 10 decimal é A em hexa. O resultado da divisão (76) é dividido, por sua vez, por 16, e dá 4, com resto de 12 (C em hexa). Finalmente, 4 dividido por 16 dá zero, com 4 de resto. Conclusão: 1226 em decimal é 4CA em hexa (note a seqüência inversa à divisão para os dígitos hexa). Para aprender melhor como funcionam todas essas conversões, digite o programa abaixo.

```
20 CLS 0
30 PRINT @11,"BIN,DEC,HEX";
40 PRINT @68,"BINARIO";
50 PRINT @196,"DECIMAL";
60 PRINT @323,"HEXADECIMAL";
70 PRINT @355,"+  +  +   = ";
80 PRINT @371,"+  +  +   = ";
90 FOR J=1 TO 15:POKE 1040+32*J
,175:NEXT
100 PRINT @450,"NUMERO HEX=
 ";
110 PRINT @227,"+    +    +    +
  +    +    + ";
120 GOTO 170
130 IN$=INKEY$:IF IN$="" THEN 1
30
140 IF IN$=" " THEN NO=NO+1:NO=
NO AND 255:GOTO 170
150 IF IN$="B" THEN NO=NO-1:NO=
NO AND 255:GOTO 170
160 GOSUB 370
170 GOSUB 190: GOSUB 270
180 GOTO 130
190 FOR X=7 TO 0 STEP-1
200 IF(NO AND 2^X)THEN N=1 ELSE
 N=0
210 PRINT @125-X*4,N;
220 IF N=1 THEN N=INT(2^X):N$=S
TR$(N):N$=MID$(N$,2,LEN(N$)-1)E
LSE N$=RIGHT$("   0",LEN(STR$(2^
X))-1)
230 PRINT @255-X*4-LEN(N$),N$;
240 NEXT
250 PRINT @279," = ";MID$(STR$(
NO)+"   ",2,3);
260 RETURN
270 FOR X=7 TO 4 STEP -1
280 PRINT @374-X*3,STR$((NO AND
 2^X)/16);
290 NEXT
300 PRINT @367,HEX$(NO/16);
310 FOR X=3 TO 0 STEP -1
320 PRINT @378-X*3,STR$(NO AND
2^X);
330 NEXT
340 PRINT @383,HEX$(NO AND 15);
350 POKE 1488,PEEK(1391):POKE 1
489,PEEK(1407)
360 RETURN
370 NU$="":PRINT @439,"?";
380 IN$=INKEY$:IF(IN$<"0" OR IN
$>"9") AND IN$<>CHR$(13) THEN G
OTO 380
390 IF IN$=CHR$(13) THEN NO=VAL
(NU$):IF NO>255 THEN 370 ELSE P
RINT @439,STRING$(5,CHR$(128));
:RETURN
400 IF IN$<>CHR$(13) AND LEN (N
U$)>2 THEN 380
410 NU$=NU$+IN$:PRINT @441,MID$
(NU$+"   ",1,3);:GOTO 380
```

```
20 CLS
25 PLOT 140,0: DRAW 0,160
30 PRINT  INVERSE 1;AT 0,8;"
BIN,DEC,HEX "
40 PRINT  INVERSE 1;AT 4,2;"
BINARIO:     "
50 PRINT  INVERSE 1;AT 9,2;"
DECIMAL:     "
60 PRINT AT 10,5;"+    +    +
 +    +    +    +"
70 PRINT  INVERSE 1;AT 17,2;"
HEXADECIMAL:"
80 PRINT AT 18,4;"+  +  +  ="
90 PRINT AT 18,20;"+  +  +  =
"
100 LET no=0
110 GOTO 150
120 LET  a$=INKEY$  :  IF a$=""
 THEN  GOTO 120
130 IF a$=" " THEN  LET no=no+
1 : IF  no=256 THEN  LET no=0
135 IF a$="b" THEN  LET no=no-
```

```
1 : IF  no=-1 THEN  LET no=255
140 IF a$="b" OR a$=" " THEN
GOTO 150
145 INPUT "?";no
150 GOSUB 170: GOSUB 250
160 GOTO 120
170 LET nu=no: LET c=128
175 FOR x=0 TO 7
180 LET n=0: IF nu>=c THEN
LET n=1 : LET nu=nu-c
190 LET c=c/2
200 PRINT AT 5,2+4*x;n
210 IF n=1 THEN  PRINT AT 10,2
+4*x;c* 2
220 IF n=0 THEN  PRINT AT 10,2
+4*x;"0  "
230 NEXT x
235 PRINT AT 13,6;"TOTAL DECIM
AL= ";n o;"  "
240 RETURN
250 LET hi=INT (no/16): LET hh
=hi
260 LET lo=(no-hi*16): LET ll=
lo: IF  lo>9 THEN  LET lo=lo+7
265 IF hi>9 THEN  LET hi=hi+7
270 LET hi=hi+48: LET lo=lo+48
280 PRINT AT 18,14;CHR$ hi;AT
18,30;  CHR$ lo
290 LET c=8
300 FOR x=0 TO 3
310 LET n=0: IF hh>=c THEN
LET n=c : LET hh=hh-c
315 LET m=0: IF ll>=c THEN
LET m=c : LET ll=ll-c
320 LET c=c/2
330 PRINT AT 18,2+x*3;n;AT 18,
18+x*3; m
340 NEXT x
400 PRINT AT 21,6;"TOTAL HEX=
";CHR$  hi;CHR$ lo
500 RETURN
```

```
 20 CLS
 30 PRINT AT 0,9;"BIN,DEC,HEX"
 40 PRINT AT 4,4;"BINARIO:"
 50 PRINT AT 9,4;"DECIMAL:"
 60 PRINT AT 10,5;"+    +    +
+    +    +    "
 70 PRINT AT 17,4;"HEXADECIMAL:
"
 80 PRINT AT 18,4;"+  +  +  ="
 90 PRINT AT 18,20;"+  +  +  ="
100 LET NO=0
110 GOTO 150
120 LET A$=INKEY$
125 IF A$="" THEN GOTO 120
130 IF A$<>"F" THEN GOTO 135
131 LET NO=NO+1
132 IF NO=256 THEN LET NO=0
135 IF A$<>"B" THEN GOTO 140
136 LET NO=NO-1
137 IF NO=-1 THEN LET NO=255
140 IF A$="B" OR A$="F" THEN GO
TO 150
145 INPUT NO
150 GOSUB 170
155 GOSUB 250
160 GOTO 120
170 LET NU=NO
171 LET C=128
175 FOR X=0 TO 7
180 LET N=0
185 IF NU>=C THEN LET N=1
186 IF NU>=C THEN LET NU=NU-C
190 LET C=C/2
200 PRINT AT 5,2+4*X;N
210 IF N=1 THEN PRINT AT 10,2+4
*X;C*2
220 IF N=0 THEN PRINT AT 10,2+4
*X;"0  "
230 NEXT X
235 PRINT AT 13,6;"TOTAL DECIMA
L= ";NO;"  "
240 RETURN
250 LET HI=INT  (NO/16)
255 LET HH=HI
260 LET LO=(NO-HI*16)
261 LET LL=LO
270 LET HI=HI+28
275 LET LO=LO+28
280 PRINT AT 18,14;CHR$ HI;AT 1
8,30;CHR$ LO
290 LET C=8
300 FOR X=0 TO 3
310 LET N=0
311 IF HH>=C THEN LET N=C
312 IF HH>=C THEN LET HH=HH-C
315 LET M=0
316 IF LL>=C THEN LET M=C
317 IF LL>=C THEN LET LL=LL-C
320 LET C=C/2
330 PRINT AT 18,2+X*3;N;AT 18,1
8+X*3;M
340 NEXT X
400 PRINT AT 21,6;"TOTAL HEX= "
;CHR$ HI;CHR$ LO
500 RETURN
```

```
5 KEY OFF
10 SCREEN0
15 COLOR 1,9
20 CLS
30 LOCATE 10,0:PRINT" BIN,DEC,H
EX"
40 LOCATE 2,4:PRINT"  BINARIO:
"
50 LOCATE 2,9:PRINT"  DECIMAL:
"
60 LOCATE 3,10:PRINT"+    +    +
  +    +    +    +"
70 LOCATE 2,17:PRINT"  HEXADECI
MAL:"
80 LOCATE 4,18:PRINT "+  +  +
="
90 LOCATE 20,18:PRINT"+  +  +
="
100 LET NO=0
110 GOTO 150
120 LET A$=INKEY$:IF A$="" THEN
 GOTO 120
130 IF A$=" " THEN LET NO=NO+1:
IF NO=256 THEN LET NO=1
135 IF A$="B" THEN LET NO=NO-1:
IF NO=-1 THEN LET NO=255
140 IF A$="B" OR A$=" " THEN GO
TO 150
145 LOCATE 30,0:INPUT NO
150 GOSUB 170:GOSUB 250
160 GOTO 120
170 LET NU=NO:LET C=128
175 FOR X=0 TO 7
180 LET N=0:IF NU>=C THEN LET N
=1:LET NU=NU-C
190 LET C=C/2
200 LOCATE 2+4*X,5:PRINT USING
"#";N;
210 IF N=1 THEN LOCATE 4*X,10:P
RINT USING "###";C*2
220 IF N=0 THEN LOCATE 4*X,10:P
RINT USING "###";0
230 NEXT X
235 LOCATE 6,13:PRINT "TOTAL DE
CIMAL=";NO;"  "
240 RETURN
250 LET HI=INT(NO/16):LET HH=HI
260 LET LO=(NO-HI*16):LET LL=LO
:IF LO>9 THEN LET LO=LO+7
265 IF HI>9 THEN HI=HI+7
270 LET HI=HI+48:LET LO=LO+48
280 LOCATE 14,18:PRINT CHR$(HI)
;
285 LOCATE 30,18:PRINT CHR$(LO)
;
290 LET C=8
300 FOR X=0 TO 3
310 LET N=0:IF HH>=C THEN LET N
=C:LET HH=HH-C
315 LET M=0:IF LL>=C THEN LET M
=C:LET LL=LL-C
320 LET C=C/2
330 LOCATE 2+X*3,18:PRINT USING
"c";N?
```

Nos micros TRS-Color, TRS-80 (com BASIC em disquete) e MSX, existem funções específicas para converter decimais em hexas. Essas funções são (*d* corresponde a um número ou expressão em decimal):

HEX$ (d)

Se você quiser incluir números em hexadecimal como parte de um programa (por exemplo, dentro de declarações **DATA**), também não há problema para muitos computadores. Ao utilizar essas constantes em hexa para cálculos subseqüentes, o computador os converterá internamente em decimal (ou, mais propriamente, em binário).

Coloque &H antes do número. Exemplo: &HF3

Coloque $ antes do número. Exemplo: $F6.

```
335 LOCATE 18+X*3,18:PRINT USIN
G"#";M;
340 NEXT LOCATE
400 LOCATE 6,21:PRINT"TOTAL HEX
= ";CHR$(HI);CHR$(LO);
500 RETURN
```

```
20  HOME
25  POKE 34,22
30  INVERSE : HTAB 14: PRINT "B
IN,DEC,HEX"
40  VTAB 6: HTAB 5: PRINT "BINA
RIO"
50  VTAB 10: HTAB 5: PRINT "DEC
IMAL"
60  VTAB 16: HTAB 5: PRINT "HEX
ADECIMAL"
70  VTAB 22: HTAB 3: PRINT "NUM
ERO HEX=        "
80  VTAB 17: PRINT "   +   +
+   =   ";: HTAB 21: PRINT "
+   +   +   =   "
90  VTAB 11: PRINT "    +    +
   +    +    +    +    +    "
110  GOTO 170
120  VTAB 23: CALL  - 958: HTAB
 30: GET IN$
130  IF IN$ =  CHR$ (32) THEN N
O = NO + 1: IF NO = 256 THEN NO
 = 0
140  IF IN$ =  CHR$ (66) THEN N
O = NO - 1: IF NO =  - 1 THEN N
O = 255
150  IF IN$ < > CHR$ (32) AND
 IN$ < > CHR$ (66) THEN  GOSU
B 410
170  GOSUB 190: GOSUB 270
180  GOTO 120
190 NU = NO:C = 128: FOR X = 7
TO 0 STEP  - 1
200 N = 0: IF NU >  = C THEN N
= 1:NU = NU - C
205 C = C / 2
210  VTAB 7: HTAB  ABS (X - 7)
* 5 + 1: PRINT "  ";N;" "
220  IF N = 1 THEN N = 2 ^ X:N$
=  STR$ (N): GOTO 230
225 N$ = "   0"
230  VTAB 11: HTAB  ABS (X - 7)
 * 5 + 1 + (3 -  LEN (N$)): PRI
NT N$
240  NEXT
250  VTAB 13: HTAB 5: PRINT "TO
TAL DECIMAL= ";NO;" ";: CALL  -
 868
260  RETURN
270 HI =  INT (NO / 16):HH = HI
280 LO = (NO - HI * 16):LL = LO
: IF LO > 9 THEN LO = LO + 7
290  IF HI > 9 THEN HI = HI + 7
300 HI = HI + 48:LO = LO + 48
310  VTAB 17: HTAB 18: PRINT  C
HR$ (HI);: HTAB 38: PRINT  CHR$
 (LO)
320 C = 8
330  FOR X = 0 TO 3
340 N = 0: IF HH >  = C THEN N
= C:HH = HH - C
350 M = 0: IF LL >  = C THEN M
= C:LL = LL - C
```

```
360 C = C / 2
370  VTAB 17: HTAB X * 4 + 2: P
RINT N;: HTAB X * 4 + 22: PRINT
 M
380  NEXT
390  VTAB 22: HTAB 15: PRINT  C
HR$ (HI); CHR$ (LO)
400  RETURN
410  VTAB 23: HTAB 30: INPUT NO
420  IF NO > 255 OR NO < 0 THEN
 410
430  NORMAL : VTAB 23: HTAB 30:
 PRINT "        ": INVERSE : RETU
RN
```

Uma vez digitado o programa, rode-o com o comando **RUN**. Aparecerão na tela três linhas: uma contendo os dígitos do número em binário; mais abaixo, a linha do número em decimal; e, por último, os dois segmentos do número em hexadecimal.

Inicialmente, todos esses números estão zerados. Ao pressionar a tecla de espaçamento (a tecla **F** na versão para o ZX-81), o número binário no topo da tela será incrementado de 1 e seus equivalentes em decimal e hexadecimal aparecerão na parte de baixo da tela. O equivalente decimal é calculado somando-se os valores (potências de 2) correspondentes às posições onde um dígito binário é igual a 1. Exemplo:

BINÁRIO:
0 0 1 0 0 1 1 0

DECIMAL:
0 + 0 + 32 + 0 + 0 + 4 + 2 + 0 = 38

POTÊNCIAS DE 2:
$2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0$

O mesmo método de conversão é utilizado para o sistema hexadecimal; apenas, neste caso, formam-se quatro grupos de quatro bits de cada vez.

Pressionando a tecla **<B>**, você diminuirá 1 do número registrado na tela. Ao mesmo tempo, se você pressionar qualquer outra tecla, além da barra de espaços ou da letra B, poderá ordenar rapidamente ao computador que converta para binário e hexadecimal um número qualquer em decimal entre 0 e 255. Ao pressionar qualquer tecla, aparecerá em um canto da tela um sinal de interrogação. Digite então o número desejado e acione **<ENTER>** ou **<RETURN>**. Ato contínuo, os números equivalentes em binário e hexa serão mostrados no vídeo.

Note que o número máximo que pode ser representado por um byte de oito bits em binário é 11111111. Isso equivale a 255 em decimal e a FF em hexadecimal. Qualquer número armazenado em um byte na memória do seu computador pode ser representado por um valor em hexa com dois dígitos.

Com o tempo, você verificará que essas conversões não são tão difíceis quanto parecem a princípio.

O sistema hexadecimal parece difícil e complicado. Não seria possível passar sem ele? Afinal, é realmente necessário aprendê-lo para operar em linguagem de máquina?

Sim. Se você tem a intenção de aprender seriamente a programar em código de máquina, um bom conhecimento sobre como trabalhar com o sistema hexadecimal é imprescindível.

Existem programas prontos para fazer a conversão automática de códigos de operação em números binários (Assembler) ou a conversão no sentido inverso (Disassembler). Além disso, eles aceitam números expressos no sistema decimal. No entanto, mesmo nesses casos, o programador precisa recorrer constantemente aos seus equivalentes hexadecimais. Muitas vezes, certas listagens emitidas por tais programas (DUMP) contêm apenas números hexadecimais. E você não saberia trabalhar com eles se não conhecesse o sistema de base 16.

Em lições posteriores, aprenderemos a fazer Assembly manualmente: o conhecimento do sistema hexadecimal será um pré-requisito para entendê-las.

Com o tempo e a prática, você verá que a dificuldade desse sistema é apenas aparente. Na verdade, ele é tão simples e fácil quanto o sistema decimal ou qualquer outro.

Como devo fazer para reconverter de hexa para decimal?

Cada dígito de um número em hexa vale 16 vezes o dígito à sua direita. Para converter um número em hexa, como F6DA, por exemplo, deve-se tomar inicialmente o primeiro dígito da direita e convertê-lo para notação decimal. No exemplo, A equivale a 10. O próximo dígito à esquerda vale 16 vezes mais; assim, convertemos D para decimal (o que dá 13), e realizamos a operação: $13 \times 16 = 208$. O próximo dígito à esquerda vale 16 vezes mais; assim, multiplicamos $6 \times 16 \times 16 = 1536$. O último dígito do exemplo precisa também ser multiplicado por 16. F vale 15, assim temos $15 \times 16 \times 16 \times 16 = 61440$. Somando todos os valores obtidos temos $10 + 208 + 1536 + 61440 = 63194$, em decimal.

Ou então use o programa listado aqui para converter o número hexa de dois em dois dígitos de cada vez, e depois multiplique o par mais à esquerda por 256.

NÚMEROS MAIORES

Como um computador de oito bits representa números maiores do que 255? Simplesmente quebrando-os em duas partes de oito bits cada e colocando-os em duas memórias adjacentes. Assim, o computador armazena qualquer número até FFFF em hexa, ou 65 535 em decimal.

O FFFF é um valor importante em computadores pessoais de oito bits, pois é o número máximo de posições endereçáveis de memória RAM.

Números maiores ainda do que esses podem ser armazenados através do mesmo estratagema: dividi-los em três, quatro ou até mais porções de oito bits.

A maneira como esses bytes são organizados depende do computador. A maior parte dos micros que usam o BASIC armazenam os bytes de menor valor (chamados LSB, ou *least significant byte*) nas memórias de endereço mais baixo, e os bytes de maior valor (MSB, ou *most significant byte*), nas memórias de endereço mais alto.

Outros computadores, como os compatíveis com o TRS-Color, fazem exatamente o contrário.

Suponhamos agora que você queira encontrar o número decimal equivalente a dois hexadecimais armazenados em compartimentos vizinhos de memória (um número maior do que FF, quebrado em dois).

Nos microcomputadores das linhas Sinclair, TRS-80, MSX e Apple II basta converter os dois hexadecimais (H1 e H2) para decimal (D) e depois fazer a seguinte operação:

Veja como o programa de conversão entre bases aparece na tela dos microcomputadores. A disposição da tela é bastante parecida a esta, nos micros de outras linhas. Agora fica mais fácil entender como os três sistemas numéricos funcionam. Quando você manda executar o programa, as três linhas são zeradas. Pressione a letra B para diminuir o valor de 1 na tela, e a linha de binário ficará totalmente cheia de 1s. Logo abaixo, a linha decimal irá completar-se com todas as potências de 2. Da direita para a esquerda você terá os valores: 1 (que é 2), 2 (que é 2), 4 (ou 2), etc. A linha hexadecimal, mais abaixo, funciona da mesma forma, só que os dois dígitos hexa são computados independentemente.

$$D = H1 - 256 + H2$$

Nos micros compatíveis com o TRS-Color, a expressão usada é a mesma, mas a ordem de H1 e H2 é inversa; ou seja, o byte de maior valor aparece antes do menor byte.

O SISTEMA OCTAL

Existem apenas quatro sistemas de numeração que podem ser utilizados com vantagem na programação e operação de computadores digitais. Você já conhece três deles: o **sistema binário**, *que é o natural para um computador digital; o* **sistema decimal**, *espontaneamente utilizado pelo ser humano; e o* **sistema hexadecimal**, *cuja base (16) é uma potência de 2. Essa correspondência facilita enormemente sua utilização pelo programador.*

O quarto sistema de numeração que pode ser usado com computadores digitais é o **sistema octal**, *ou seja, de base 8, que é também uma potência do número 2. Apesar disso, o octal está entrando em desuso, pois não foi universalmente adotado pelos fabricantes de microcomputadores.*

O octal tem 8 dígitos: 0, 1, 2, 3, 4, 5, 6, 7 e 8. O sistema de conversão de um número em octal para decimal é semelhante ao usado com o hexadecimal, só que o multiplicador é 8. Por exemplo:

$$423 = 4 \times 64 + 2 \times 8 + 3 = 275$$

Alguns computadores pessoais permitem representar constantes em octal **(&0)**, *ou têm funções de conversão* **(OCT$)**.

MARQUE O TEMPO E OS PONTOS

- O JOGO DO CAMPO MINADO
- APRENDA A MONTAR UM PLACAR
- COMO INCORPORAR AO JOGO UM MEDIDOR DE TEMPO

Todo jogo de ação realiza algum tipo de contagem de pontos ou de tempo, de modo a tornar a competição mais emocionante. Nesta lição você aprenderá como fazer isso com a ajuda de programas bem simples.

Quase todos os jogos de computadores precisam de alguma forma de marcação de pontos ou de cronometragem — ou ambos. Sem eles você não poderá saber se está se saindo bem no jogo, se está melhorando, ou quem é o melhor jogador. Além disso, são esses elementos de medida de desempenho que dão graça aos jogos de computador.

Seria possível, evidentemente, obrigar alguém a sentar-se ao seu lado e ficar contando os disparos que você faz. A melhor solução, no entanto, é programar o computador para fazer isso por você. Umas poucas linhas a mais no programa darão cabo dessa tarefa, não importa a complexidade do esquema de marcação de pontos.

Da mesma forma, não existe necessidade de recorrer a um cronômetro para marcar o tempo. Como se viu nos programas para o jogo de labirinto (pág. 46 a 52), quase todos os tipos de computador têm um relógio embutido, que pode ser utilizado para melhorar seus jogos (a exceção é o Apple II, que precisa de uma placa especial).

CAMPO MINADO

Para que você possa ver como montar esquemas práticos de contagem e cronometragem, apresentamos a seguir um jogo adequado a todas as máquinas, com exceção das compatíveis com o ZX81, e ao qual acrescentaremos progressivamente todas as rotinas necessárias. Os esquemas são simples e podem ser colocados também em outros jogos.

O jogo é chamado Campo Minado, e nele você é um comandante de tanque cuja missão é resgatar os pára-quedistas que estão pousando imprudentemente em um campo minado. Por onde quer

que passe, o tanque corre o risco de detonar uma mina escondida ao acaso pelo computador. Assim como em campo de batalha real, as minas são invisíveis; portanto, mova-se com cuidado!

O tanque (apenas um caractere #, infelizmente, pois você ainda não aprendeu tudo sobre combinar gráficos e movimentos em um programa BASIC) executa movimentos por meio das seguintes teclas: Z para a esquerda, X para a direita, P para cima e L para baixo. O âmago do programa está na rotina de movimentação de um caractere na tela estudada numa lição anterior.

Quando você digitar e rodar esta seção do jogo verá que ele ainda não está completo, pois nada acontece depois que o pára-quedista é resgatado. Apenas o tanque continua a vagar, sem rumo, através da tela. O programa deve ser interrompido pressionando-se as teclas <**BREAK**>, <**ESCAPE**> ou <**STOP**>, ou você terá que esperar até que o tanque detone uma mina escondida ao acaso. Nesse momento, o jogo terminará. Mas não se preocupe, essa situação melhorará tão logo você acrescente as rotinas de contagem e cronometragem que se seguem.

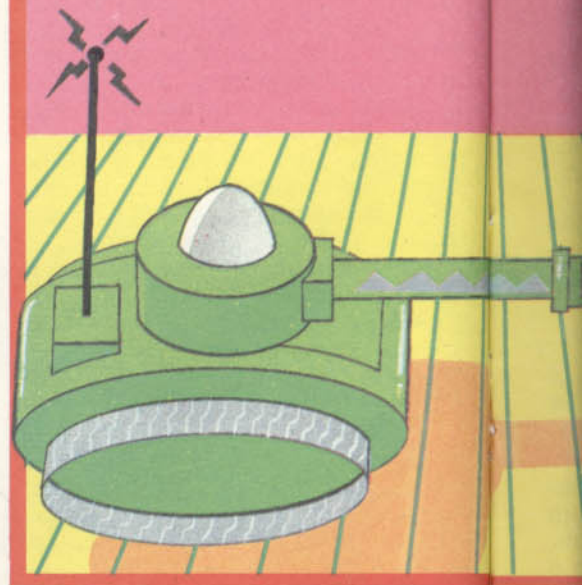
```
50 LET PO=210
60 CLS
70 PRINT @256,STRING$(32,"-")
90 LET X=RND(256)-1
110 IF X<>PO THEN PRINT @X,"O";
ELSE GOTO 90
120 PRINT @PO,"#";
130 LET LP=PO
140 IF PEEK(340)=247 THEN LET P
O=PO-1:GOTO 190
150 IF PEEK(338)=247 THEN LET P
O=PO+1:GOTO 210
160 IF PEEK(338)=251 THEN LET P
O=PO-32:GOTO 220
170 IF PEEK(342)=253 THEN LET P
O=PO+32:GOTO 220
180 GOTO 140
190 IF (LP AND 31)=0 THEN LET P
O=LP
200 GOTO 220
210 IF (PO AND 31)=0 THEN LET P
O=LP
220 IF PO>255 OR PO<0 THEN LET
PO=LP:GOTO 140
230 PRINT @LP," ";
240 PRINT @PO,"#";
250 LET M=RND(256)-1
270 IF M=PO THEN PRINT @PO," ":
PRINT @130,"BOOM!!! VOCE ACERTO
U UMA MINA!":STOP
310 GOTO 130
```

O movimento do tanque pela tela é controlado pelas linhas 140 a 170, que verificam quais foram as teclas pressionadas (veja a pág. 33, se não se lembra de como isso é feito). A linha 220 testa IF PO>255... para manter o tanque no alto da parte central da tela. Ora, a última locação de tela no alto da parte central do vídeo é a 255; assim o **IF...** impede que o tanque avance abaixo da linha pontilhada, desenhada pela linha 70.

Esta última traça a linha através da tela, utilizando uma nova função — **STRING$**. O **STRING$(32,"—")**, como aparece na linha 70, diz ao TRS-Color para desenhar um cordão (*string*, em inglês) de 32 travessões (se você quisesse dez pontos de interrogação deveria utilizar **STRING$(10,"?")**, e assim por diante). O lugar em que o pára-quedista cai é escolhido por um número aleatório na linha 90, e a linha 110 exibe o pára-quedas na tela se a posição escolhida já não estiver ocupada pelo tanque. Se estiver, a linha 90 escolherá outra posição.

Finalmente, a posição da mina escondida é escolhida pela linha 250. A linha 270 checa então se o tanque e a mina estão na mesma locação de tela. Se estiverem, a mensagem de explosão será exibida, e o programa parará.

```
50 LET tx=16: LET ty=5
60 CLS
70 PRINT AT 11,0;"-----------
---------------------"
90 LET px=INT (RND*30)+1
100 LET py=INT (RND*10)
110 IF px=tx AND py=ty THEN
GOTO 90
120 PRINT AT py,px;"O";AT ty,
tx;"#"
130 LET txx=tx: LET tyy=ty
140 LET a$=INKEY$
145 IF a$="w" THEN  LET ty=ty-
1
150 IF a$="z" THEN  LET ty=ty+
1
160 IF a$="a" THEN  LET tx=tx-
1
170 IF a$="s" THEN  LET tx=tx+
1
190 IF ty<0 OR ty>10 THEN  LET
ty=tyy
200 IF tx<0 OR tx>30 THEN  LET
tx=txx
230 PRINT AT tyy,txx;" "
240 PRINT AT ty,tx;"#"
250 LET mx=INT (RND*30)+1
260 LET my=INT (RND*10)
270 IF mx=tx AND my=ty THEN
PRINT AT  my,mx;" ": PRINT AT
8,0;"BOOM!!! - Voce acertou um
a mina!":   GOTO 330
310 GOTO 130
```

No programa do Spectrum, a tela é dividida ao meio pela linha 70, que imprime uma série de 32 travessões ao longo do vídeo. As linhas 145 a 170 controlam os movimentos do tanque. Uma explicação completa desse processo está na página 31.

As linhas 190 e 200 impedem que o tanque saia da área situada acima da linha (pontilhada) traçada pela linha 70, e impede que ele desapareça da tela. A posição de queda dos pára-quedistas é

escolhida aleatoriamente pelas linhas 90 e 100. A linha 110 verifica se o tanque e um dos pára-quedistas ocupam a mesma locação de tela. Se isso ocorrer, uma nova posição de queda será escolhida, indo-se para a linha 90. As linhas 250 e 260 escolhem um lugar para a mina. A linha 270 compara a posição da mina com a do tanque. Se eles estiverem no mesmo lugar, o tanque será exibido na tela e aparecerá uma mensagem de explosão. Neste caso, o programa parará.

```
50 LET PO=220
60 CLS
70 LOCATE 0,12:PRINT STRING$(39
,"_")
90 LET X=INT(RND(1)*480)
110 IF X<>PO THEN VPOKE BASE(0)
+X,ASC("O") ELSE GOTO 90
120 VPOKE BASE(0)+PO,ASC("‡")
130 LET LP=PO
135 LET K$=INKEY$:IF K$="" THEN
 GOTO 135
140 IF K$=CHR$(29) THEN LET PO=
PO-1:GOTO 190
150 IF K$=CHR$(28) THEN LET PO=
PO+1:GOTO 210
160 IF K$=CHR$(30) THEN LET PO=
PO-40:GOTO 220
170 IF K$=CHR$(31) THEN LET PO=
PO+40:GOTO 220
180 GOTO 140
190 IF (LP/40-INT(LP/40))=0 THE
N LET PO=LP
200 GOTO 220
210 IF (PO/40-INT(PO/40))=0 THE
N LET PO=LP
220 IF PO>479 OR PO<0 THEN PO=L
P:GOTO 130
230 VPOKE BASE(0)+LP,ASC(" ")
240 VPOKE BASE(0)+PO,ASC("‡")
250 LET M=INT(RND(1)*480)
270 IF M=PO THEN VPOKE BASE(0)+
PO,ASC(" "):LOCATE 5,4:PRINT "B
OOM!!-Você acertou numa mina":E
ND
310 GOTO 130
```

O movimento do tanque pela tela é, como no caso anterior, controlado pelas linhas 140 a 170, que verificam quais foram as teclas pressionadas. A linha 220 testa **IF PO>479...** a fim de restringir o campo de ação do tanque à parte alta do meio do vídeo. Agora, porém, a última locação de tela nessa posição é a 479; assim, **O IF...** impede que ela avance abaixo da linha pontilhada desenhada pela linha 70.

Também como no caso anterior a função **STRING$** é empregada pela linha 70 para traçar a linha ao longo da tela. O **STRING$(39, "—")**, como aparece na linha 70, diz simplesmente ao MSX para desenhar um cordão de 39 travessões (veja na pág. anterior, abaixo do programa para o TRS80 e o TRS-Color, como fazer para obter dez pontos de interrogação).

O lugar em que o pára-quedista cai é escolhido por um número na linha 90; a linha 110 mostra o pára-quedas na tela se a posição escolhida já não estiver ocupada pelo tanque. Se estiver, a linha 90 escolherá outra posição. Não se preocupe em entender, por enquanto, o comando **VPOKE BASE**, na linha 110. Ele serve para colocar um caractere comum (ou gráfico) em uma certa posição da tela, e é usado aqui por ser mais rápido que o **LOCATE**.

A linha 250, escolhe a posição da mina escondida. A linha 270 verifica se o tanque e a mina estão na mesma locação da tela. Sendo positivo, a mensagem de explosão é exibida, e o programa termina..

```
50   LET TX = 20: LET TY = 10
60   HOME
70   FOR I = 1 TO 40
74   HTAB I: VTAB 13: PRINT "-";
78   NEXT I
90   LET PX =  INT ( RND (1) * 4
0) + 1
100  LET PY =  INT ( RND (1) *
12) + 1
110  IF PX <  > TX AND PY <  >
TY THEN  HTAB PX: VTAB PY: PRIN
T "O";: GOTO 120
111  GOTO 90
120  HTAB TX: VTAB TY: PRINT "‡
";
130  LET LX = TX: LET LY = TY
140  LET K =  PEEK ( - 16384):
POKE  - 16368,0
150  IF K = 208 THEN  LET TY =
TY - 1
160  IF K = 204 THEN  LET TY =
TY + 1
170  IF K = 218 THEN  LET TX =
TX - 1
175  IF K = 216 THEN  LET TX =
TX + 1
190  IF TX < 1 OR TX > 40 THEN
 LET TX = LX
200  IF TY < 1 OR TY > 12 THEN
 LET TY = LY
230  HTAB LX: VTAB LY: PRINT "
";
240  HTAB TX: VTAB TY: PRINT "‡
";
250  LET MX =  INT ( RND (1) *
40) + 1
260  LET MY =  INT ( RND (1) *
12) + 1
270  IF MX = TX AND MY = TY THE
N  HTAB MX: VTAB MY: PRINT " ";
: HTAB 7: VTAB 7: PRINT "BOOM!!
-Voce acertou uma mina";: GOTO
330
310  GOTO 130
```

O movimento do tanque pela tela é controlado pelas linhas 140 a 170, que checam as teclas pressionadas (lembre-se da explicação da pág. 33). Os comandos **PEEK** e **POKE** da linha 140 são necessários porque o Apple não tem uma função em BASIC, como o **INKEY$**, que serve para fazer a varredura do teclado. A fim de restringir as manobras do tanque, as linhas 190 e 200 testam se as suas coordenadas (TX e TY) não excedem 40 e 12 posições, respectivamen-

te (margem direita da tela e posição de linha acima da linha pontilhada).

As linhas 70 a 78 desenham uma linha pontilhada ao longo da tela.

Um número aleatório nas linhas 90 e 100 escolhe o lugar em que o pára-quedista cai; a linha 110 exibe o pára-quedas na tela se a posição escolhida já não estiver ocupada pelo tanque. Se estiver, então a linha 90 escolherá uma outra posição. Neste caso, a posição da mina escondida é escolhida pelas linhas 250 e 260. A linha 270 então verifica se o tanque e a mina estão na mesma locação da tela. Se eles estiverem, será exibida uma mensagem de explosão, e o programa se deterá.

MARQUE PONTOS

Em jogos do tipo videogame, a contagem normalmente é aumentada quando as posições de tela de dois objetos são iguais. Os objetos podem ser um míssil e, o seu alvo, o "come-come" e um monstrinho, um tanque e um pára-quedista, um cavalo e o poste de chegada, ou qualquer outra coisa.

Assim, acrescente estas linhas ao seu programa, para ver como um mecanismo de contagem funciona na prática.

```
40 LET S=0
280 IF X=PO THEN LET S=S+1:GOTO
 90
330 PRINT @295,S;"PARAQUEDISTAS
";
```

```
 40 LET s=0
280 IF px=tx AND py=ty THEN
 LET s=s+1 : GOTO 90
330 PRINT AT 14,8;s;" PARAQUED
ISTAS"
```

```
40 LET S=0
280 IF X=PO THEN LET S=S+1:GOTO
 90
330 LOCATE 8,16:PRINT S;" paraq
uedistas"
```

```
40  LET S = 0
280  IF PX = TX AND PY = TY THE
N  LET S = S + 1: GOTO 90
330  HTAB 12: VTAB 15: PRINT S;
" PARAQUEDISTAS";
```

Você precisará substituir o **STOP** ou **END** na linha 270 por **GOTO 330**. A linha 270 deverá ficar assim, agora:

```
270 IF M=PO THEN PRINT @PO," ":
PRINT @
130,"BOOM!!! VOCE ACERTOU UMA M
INA!":G OTO 330
```

```
270 IF mx=tx AND my=ty THEN
PRINT AT my,mx;"!": PRINT AT 8
,3;"BOOM!!! - VOCE ACERTOU UMA
 MINA!": GOTO 310
```

```
270 IF M=PO THEN VPOKE BASE(0)+
PO,ASC(" "):LOCATE 5,4:PRINT "B
OOM!!-Você acertou numa mina":G
OTO 330
```

```
270  IF MX = TX AND MY = TY THE
N  HTAB MX: VTAB MY: PRINT " ";
: HTAB 7: VTAB 7: PRINT "BOOM!!
-VOCE ACERTOU NUMA MINA";: GOTO
 330
```

A linha 280 é a mais importante. Ela checa se o tanque e o pára-quedista estão na mesma locação da tela. Se estiverem, então o placar será incrementado de 1 ponto.

A linha 40 volta o placar para zero, antes de o jogo começar, e a linha 330 exibe a contagem. Ao modificar-se o final da linha 270, o computador exibe a contagem depois que a mina tiver sido detonada.

O jogo funciona assim: o jogador recebe continuamente pára-quedistas para resgatar. Quando um é resgatado, outro pára-quedista cai do céu.

O jogo termina quando o tanque em movimento passa sobre a mina, provocando uma explosão.

RECORDES

Acrescentar um dispositivo de registro do recorde de pontos no jogo não é difícil. Tudo o que você precisa é de uma variável para armazenar o recorde, e algum modo de atualizá-la assim que este for quebrado. Fora isso, é necessária ainda uma rotina de exibição do recorde. Estas são as linhas que você deve acrescentar para aumentar a capacidade de registrar e exibir recordes no jogo "Campo Minado":

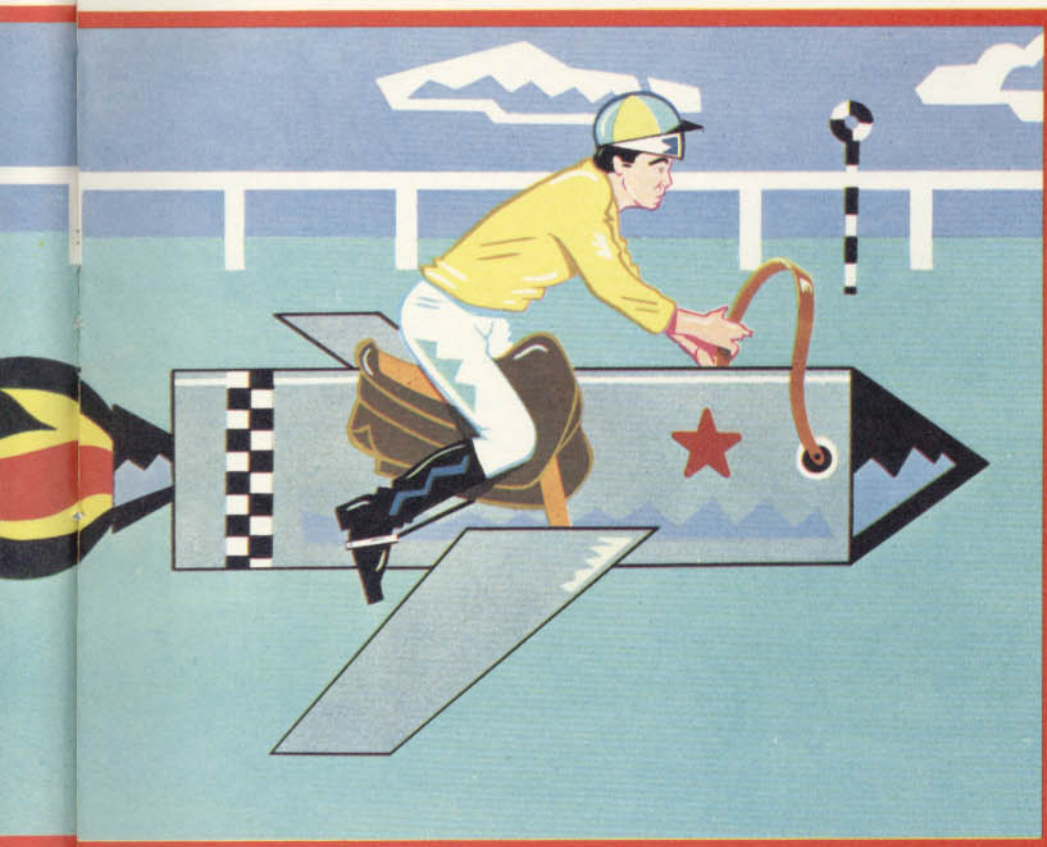
```
30 LET HS=0
350 IF S>HS THEN LET HS=S
370 PRINT @424,"RECORDE=";HS;
```

```
 30 LET hs=0
350 IF s>hs THEN  LET hs=s
370 PRINT AT 18,8;"RECORDE=";hs
```

```
30 LET HS=0
350 IF S>HS THEN LET HS=S
370 LOCATE 12,20:PRINT "Recorde
: ";HS
```

```
30  LET HS = 0
350  IF S > HS THEN  LET HS = S
370  HTAB 12: VTAB 20: PRINT "R
ecorde=";HS;
```

Primeiro o programa deve introduzir a variável contendo o recorde com um valor o mais baixo possível. Assim, a linha 30 iguala HS a zero. Quando o jogo termina, a linha 350 compara o último escore (S) com o recorde (HS). Se o escore for maior do que o recorde, então HS será atualizada, recebendo o valor em S. Finalmente, a linha 370 exibe o recorde na tela.

Pode-se pensar que essas linhas serão suficientes para incorporar sempre novos recordes ao jogo. Infelizmente isso

não é verdade. Sempre que o programa é rodado, o computador zera automaticamente o valor de HS registrado no jogo anterior, assim como os valores de todas as outras variáveis. Para manter o valor de HS entre uma partida e outra do jogo, você precisa acrescentar linhas de programa para perguntar ao jogador se ele quer jogar outra vez, como as que foram usadas em jogos anteriores.

```
390 FOR F=1 TO 1000:NEXT F
400 PRINT @130
410 PRINT @135,"OUTRA VEZ(S/N)?
"
420 LET K$=INKEY:IF K$="" THEN
GOTO 420
430 IF K$="S" THEN GOTO 40
440 IF K$="N" THEN END
450 GOTO 420
```

```
390 PAUSE 100
400 PRINT AT 8,3;TAB 31
410 PRINT AT 8,7;"Quer jogar d
e novo (S/N)?"
420 LET a$=INKEY$
430 IF a$="s" THEN  GOTO 40
440 IF a$="n" THEN  STOP
450 GOTO 420
```

```
390 FOR F=1 TO 1000:NEXT F
400 LOCATE 0,4:PRINT STRING$(39
," ")
410 PRINT TAB(10);"Outra vez? (
S/N)"
420 LET K$=INKEY$:IF K$="" THEN
 GOTO 420
430 IF K$="S" OR K$="s" THEN GO
TO 40
440 IF K$="N" OR K$="n" THEN EN
D
450 GOTO 420
```

```
390   FOR F = 1 TO 1000: NEXT F
400   FOR I = 1 TO 40: HTAB I: V
TAB 7: PRINT " ";: NEXT I
410   HTAB 11: VTAB 8: PRINT "Ou
tra vez? (S/N)";
420   GET A$
430   IF A$ = "S" THEN  GOTO 40
440   IF A$ = "N" THEN  END
450   GOTO 420
```

Isto é o que a linha extra faz:

Cria-se um pequeno retardo de tempo, introduzido pelo laço **FOR...NEXT** na linha 390, nos programas para todos os computadores, exceto para o Spectrum, que usa o comando **PAUSE**. A linha 400 limpa uma parte da tela, de modo a prepará-la para que a mensagem "MAIS UMA VEZ? (S/N)" seja impressa pela linha 410.

A rotina que pergunta ao jogador se quer jogar de novo e colhe a sua resposta está nas linhas 410 a 450 (a linha 430 reiniciará o programa na linha 40 se S for pressionado, e a linha 440 parará o programa se N for pressionado). A linha 450 assegura que qualquer outra tecla será ignorada. Como agora não existe necessidade de digitar **RUN** cada vez que você desejar jogar de novo, o valor de HS é preservado.

COMO FAZER A CRONOMETRAGEM

Do jeito que está, o jogo depende muito da sorte — o jogador simplesmente continua andando com o tanque, até encontrar uma mina escondida.

Podemos, entretanto, introduzir novos elementos no jogo, transformando-o, se quisermos, em uma corrida contra o relógio. Um desses elementos poderia ser, por exemplo, um dispositivo interno para marcar o tempo gasto em resgatar dez pára-quedistas.

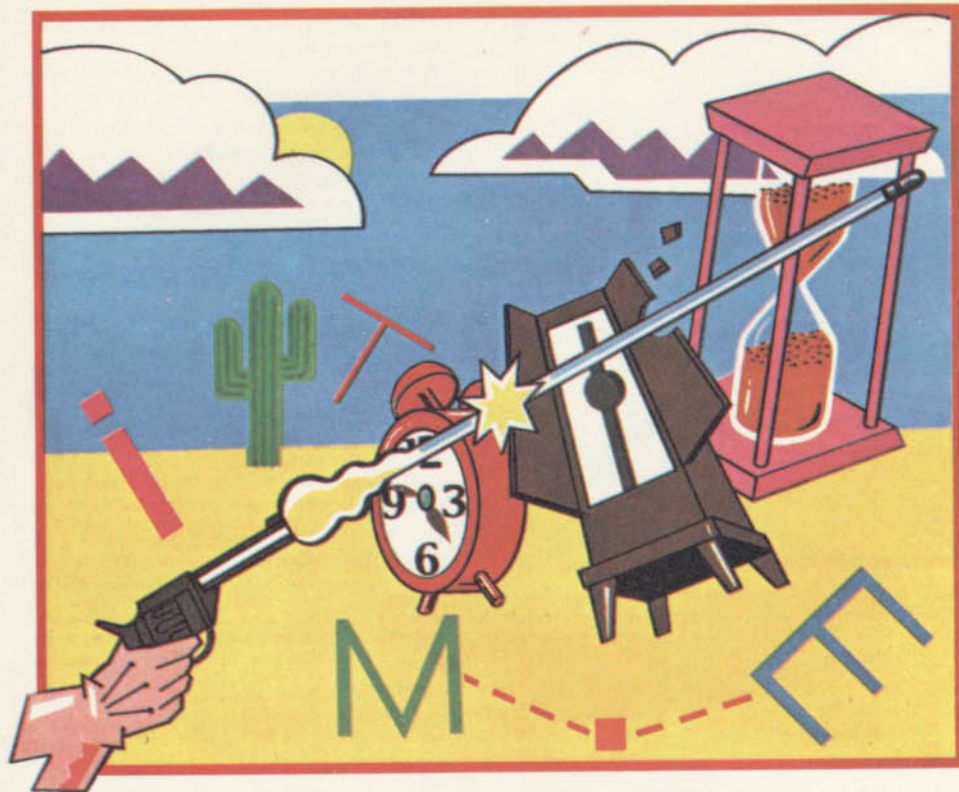
```
80 TIMER=0
290 IF S<10 AND X=PO THEN GOTO
90
300 IF S=10 THEN GOTO 320
320 LET T=TIMER
330 PRINT @295,S;"PARAQUEDISTAS
";
340 IF S=10 THEN PRINT @327,"EM
";T/50;" SEGUNDOS"
```

```
 10 LET t=0
 80 POKE 23672,0: POKE 23673,0
290 IF s<10 AND px=tx AND py=t
y THEN  GOTO 90
300 IF s=10 THEN  GOTO 320
320 LET t=PEEK 23672+256*PEEK
23673
340 IF s=10 THEN  PRINT AT 16,
8;"EM ";t/50;" SEGUNDOS"
```

```
80 TIME=0
290 IF S<10 AND X=PO THEN GOTO
90
300 IF S=10 THEN GOTO 320
320 LET T=TIME
340 IF S=10 THEN LOCATE 10,18:P
RINT "em ";T/50;" segundos"
```

O relógio existente em cada um dos tipos de micros acima corre durante todo o tempo em que o computador estiver ligado.

Para iniciar a cronometragem do jogo, você precisa apenas zerar uma va-

riável de tempo. A linha que coloca em zero o cronômetro é a linha 80 — você notará que para zerar o cronômetro do TRS-Color e do MSX basta fazer **TIMER** = *O*. No Spectrum é mais complicado: a linha do programa zera duas locações específicas na memória de trabalho do micro, através de **POKE 23 672,0** e **POKE 23 673,0**.

O relógio é "parado" pela linha 320. Na verdade, ele não se detém; assim, você simplesmente instrui a máquina a "lembrar-se" do valor de tempo marcado em um determinado momento — quando dois objetos coincidem na tela, por exemplo.

Nesses programas, a variável que contém a leitura do relógio é chamada de **T** — no MSX e no TRS-Color, isso é feito com a linha **LET T = TIMER**, e no Spectrum, com **LET T = PEEK 23672 + 256*PEEK 23673**.

O comando **PEEK**, como aprenderemos em uma lição posterior, "olha" os valores numéricos contidos nas mesmas locações de memória zeradas no início do programa.

A locação 23672, como qualquer locação de memória em um micro de 8 bits, pode conter números inteiros entre 0 e 255. A locação 23673 é incrementada de 1, toda vez que esse número for excedido na locação 23672.

Assim, para calcular o valor total do tempo em que o cronômetro do Spectrum esteve correndo, você deve multiplicar o valor da locação 23673 por 256 e acrescentar o valor armazenado na locação 23672. Esta é a razão da expressão numérica na linha 90, na versão para o Spectrum.

O relógio deve ser parado quando o jogador tiver resgatado dez pára-quedistas; assim, a linha 300 checa se este valor foi atingido e salta para a linha 320, que "pára" o cronômetro.

Se o total de resgatados for menor que dez, a linha 290 manda outro pára-quedista para ser salvo pelo jogador.

A linha 340 só exibirá o tempo total de jogo se os dez pára-quedistas tiverem sido recuperados. A leitura do cronômetro é dividida por 50 no programa. Assim, o tempo exibido será expresso, aproximadamente, em segundos.

```
80  LET T = 0
290  IF S < 0 AND PX = TX AND P
Y = TY THEN  GOTO 90
300  IF S = 10 THEN  GOTO 330
340  IF S = 10 THEN  HTAB 12: V
TAB 16: PRINT "em ";T / 6;" seg
undos";
```

A contagem de tempo na versão do programa para os microcomputadores da linha Apple é feita de forma inteiramente diferente das outras máquinas, pois eles não têm relógio interno.

Neste caso, definimos uma variável denominada **TIME**, que conterá o valor cronometrado, e uma variável **T**, que é incrementada de 1 a cada ciclo de varredura do teclado. Este é um truque muito fácil de ser implementado, e que serve para qualquer computador que não tenha relógio interno.

A variável relógio é zerada na linha 80 e recebe o valor do tempo gasto na linha 320. Para calcular o valor do tempo em segundos, divide-se o conteúdo de **T** por 6.

RECORDE DE TEMPO

Da mesma forma que adicionamos o registro de recordes de pontos ao programa anterior, poderíamos agora acrescentar um registro do recorde de tempo (no caso, o menor tempo gasto para salvar dez pára-quedistas). O princípio geral de registro de recordes de tempo pode ser aplicado em muitos outros jogos.

Estas são as linhas a serem acrescentadas ao programa:

```
20 LET LT=999999
360 IF T<LT AND S=10 THEN LET L
T=T
380 PRINT @452,"MENOR TEMPO=";L
T/50;" SEGUNDOS"
```

```
 20 LET lt=999999
360 IF t<lt AND s=10 THEN  LET
 lt=t
380 PRINT AT 21,4;"MENOR TEMPO
=";lt/50;" SEGUNDOS"
```

```
20 LET LT=999999!
360 IF T<LT AND S=10 THEN LET L
T=T
380 LOCATE 6,22:PRINT "Melhor t
empo: ";LT/50;" segundos"
```

```
20  LET LT = 99999
360  IF S = 10 AND T < LT THEN
LT = T
380  HTAB 12: VTAB 21: PRINT "e
m ";LT / 6;" segundos";
```

Inicialmente, devemos inicializar a variável usada para armazenar o menor tempo (**LT**). Esse valor, ao contrário do usado para introduzir a variável de recorde de pontos, deve ser bem alto, no começo. A linha 20, incorporada ao programa, iguala a variável **LT** a 999999.

A linha 360 compara o último tem-

po de jogo registrado com o menor tempo, armazenado em **LT**. Se aquele for menor ainda do que o último recorde de tempo, passará a ser o novo recorde, transferindo-se o seu valor para **LT**.

Finalmente, a linha 380 exibe o recorde de tempo, em segundos. O valor da variável **LT** é dividido por 50 ou por 100, para que isto se torne possível.

TEMPO DE REAÇÃO

Até agora, você viu como medir o tempo com auxílio do relógio interno do computador, dentro de um programa que checa, por exemplo, as posições de dois objetos na tela.

Outra aplicação interessante para esse modo de aferição de tempo é utilizar o teclado para medir a velocidade de reação do jogador a algum tipo de ação contrária.

Você já sabe como utilizar a função **INKEY$** (em caso de dúvida, veja na pág. 28). Ela pode servir não só para controlar a movimentação de objetos na tela, mas também para iniciar e parar contagens de tempo.

Eis aqui um jogo da variedade "rápido-no-gatilho", que ilustra muito bem esse tipo de aplicação. O programa mostra na tela: "SAQUE!", e o jogador deve reagir o mais rapidamente que puder, pressionando qualquer tecla no teclado.

```
20 CLS
30 LET N=RND(900)
40 FOR F=0 TO N
50 NEXT F
60 PRINT @269,"SAQUE!!"
70 TIMER=0
80 IF INKEY$=""THEN GOTO 80
90 LET T=TIMER
100 PRINT @269,"BANG!!!"
110 FOR F=1 TO 300
120 NEXT F
130 LET M=RND(25)
140 IF T<M THEN PRINT @264,"VOC
E SOBREVIVEU"
150 IF T>M THEN PRINT @266,"VOC
E ESTA MORTO"
160 IF T=M THEN PRINT @264,"VOC
ES ESTAO AMBOS MORTOS"
```

```
20 CLS
30 LET N=INT (RND*400)+1
40 PAUSE n
60 PRINT AT 11,14;"SAQUE!!"
70 POKE 23672,0: POKE 23673,0
80 IF INKEY$="" THEN  GOTO 80
90 LET T=PEEK 23672+256*PEEK
23673
100 PRINT AT 11,14;"BANG!!"
110 PAUSE 50
130 LET m=INT (RND*35)+1
140 IF t<m THEN  PRINT AT 11,9
;"VOCE SOBREVIVEU"
150 IF t>m THEN  PRINT AT 11,9
;"VOCE ESTA MORTO"
160 IF t=m THEN  PRINT AT 11,9
;"VOCES ESTAO AMBOS MORTOS"
```

```
20 CLS
30 LET N=RND(1)*900
40 FOR F=1 TO N
50 NEXT F
60 LOCATE 17,11:PRINT "SAQUE!"
70 TIME=0
80 IF INKEY$="" THEN 80
90 LET T=TIME/6
100 LOCATE 17,11:PRINT "BANG!!"
110 FOR F=1 TO 300
120 NEXT F
130 LET M=RND(1)*25
140 IF T<M THEN LOCATE 13,11:PR
INT "Você sobreviveu"
150 IF T>M THEN LOCATE 13,11:PR
INT "Você está morto"
160 IF T=M THEN LOCATE 13,11:PR
INT "Ambos morreram"
```

```
20   HOME
30   LET N =  RND (1) * 2000
40   FOR F = 1 TO N
50   NEXT F
60   HTAB 17: VTAB 13: PRINT "SA
QUE!!"
70   LET T = 0
80 X =  PEEK ( - 16384): POKE
- 16368,0
81   IF X > 127 THEN  GOTO 100
85   LET T = T + 1
90   GOTO 80
100   HTAB 17: VTAB 13: PRINT "B
ANG!!"
110   FOR F = 1 TO 1000
120   NEXT F
130   LET M =  RND (1) * 20
140   IF T < M THEN  HTAB 12: VT
AB 13: PRINT "VOCE SOBREVIVEU"
150   IF T > M THEN  HTAB 12: VT
AB 13: PRINT "VOCE ESTA MORTO"
160   IF T = M THEN  HTAB 12: VT
AB 13: PRINT "AMBOS MORRERAM "
```

O programa é muito simples. Após a linha 20 ter limpado a tela, uma pausa aleatória é introduzida pelas linhas 30 a 50. A linha 60 exibe "SAQUE!", e o cronômetro é acionado imediatamente pela linha 70. A linha 80 faz com que a máquina espere até que uma tecla seja pressionada. A linha é exatamente igual à que foi utilizada nos programas de jogos com controle pelo teclado (veja pág. 28).

Assim que qualquer tecla tenha sido pressionada, o programa continua para a linha 90, que pára efetivamente o cronômetro, armazenando o seu valor naquele momento na variável **T**. A linha 100 exibe "BANG!!".

Existe uma pausa introduzida pelas linhas 110 e 120 (a linha 110 somente no Spectrum) antes que um novo retardo inicial de tempo seja sorteado pelo programa. Isto é feito pela linha 130 do programa.

Em seguida, o computador sorteia uma variável aleatória **M**, que conterá o tempo levado pelo computador para "sacar" sua arma. As linhas 140 a 160 comparam os valores **T** (do jogador) e **M** (da máquina) e declaram quem foi o vencedor do duelo (ou se ambos os duelistas morreram!).

Existe algum limite máximo para o período de medida de tempo?

Sim, existe — embora normalmente seja tão longo que, na prática, não faz diferença. O relógio interno da maioria dos computadores domésticos corre a velocidades semelhantes. O fator limitante, portanto, é o número máximo de impulsos de relógio que o computador pode armazenar. As linhas TRS-Color e MSX utilizam dois bytes; por isso podem contar até 65 535 impulsos de relógio, um a cada 1/50 de segundo. Isso dá uma capacidade máxima de temporização de cerca de 22 minutos. Já os micros da linha Spectrum usam 3 bytes, o que lhes dá cerca de *quatro* dias de medida contínua de tempo!

Como funciona o cronômetro interno de um microcomputador?

Todo microcomputador tem um dispositivo interno para sincronização das atividades da Unidade Central de Processamento, que é chamado de relógio ou *clock*. Esse relógio interno tem por função gerar pulsos elétricos repetidos a uma freqüência constante para cada micro: essa freqüência é chamada de *velocidade de relógio*, e seu valor varia conforme a marca do computador.

Por exemplo, o Apple II tem relógio de I MHz (megahertz, ou seja, um milhão de "batidas" de relógio por segundo); o TRS-80 tem 2,7 MHz, e assim por diante. Mas esse temporizador interno não pode ser aproveitado para medir o tempo como um verdadeiro cronômetro.

ORGANIZE AS SUAS COLEÇÕES (1)

Seu caderninho de endereços está uma bagunça? Sua coleção de discos tem mais problemas do que discos? Eis aqui um programa que o ajudará a colocar ordem nesse caos.

As pessoas que nunca utilizaram um computador pessoal sempre perguntam que tipo de aplicações domésticas ele poderia ter. As respostas dadas, porém, raramente são convincentes.

Neste artigo apresentamos um programa que pode ser útil para muitas pessoas. Ele é um programa de *banco de dados*, ou seja, de organização e recuperação de qualquer tipo de informação que possa ser fichada. É um sistema de arquivamento tão flexível que pode encontrar dezenas de aplicações no seu dia-a-dia. Ele é muito útil, por exemplo, para armazenar os nomes e os endereços de amigos ou membros de um clube, ou tomar notas dos aniversários da família e dos amigos, ou armazenar detalhes sobre suas coleções de moedas, borboletas, discos ou receitas, ou até mesmo organizar melhor a sua crescente coleção de jogos de computador.

O único limite para o que se pode fazer com esse programa está no tamanho da memória RAM do seu micro. Na maioria das vezes, o mínimo necessário para aplicações práticas é uma máquina com 32K.

Seja como for, é necessário lembrar sempre o seguinte: como as memórias dos computadores domésticos são pequenas, em comparação com as das máquinas profissionais, quanto menos volumosa a informação a ser mantida pelo computador, melhor.

O MENU PRINCIPAL

Assim que você rodar o programa pela primeira vez, através do comando **RUN**, o menu principal será imediatamente impresso na tela. Ele consiste numa lista de coisas que se pode fazer com o arquivo, como "entrar um registro", por exemplo, ou "pesquisar o arquivo".

ABRA SEU ARQUIVO

Os computadores necessitam de detalhes precisos do que se quer, antes de qualquer operação. Para abrir um novo arquivo é necessário primeiro dizer ao computador o número de registros que se quer e a extensão máxima que cada registro pode ter. "ABRIR UM ARQUIVO" é a opção 1 do menu principal; assim, para selecioná-la, você deve pressionar a tecla 1: as palavras "VOCÊ TEM CERTEZA?" aparecerão na tela. Essa reação da máquina é uma precaução contra o pressionamento acidental da tecla 1, pois chamar a rotina de "ABRIR UM ARQUIVO" quando o sistema de arquivamento já estiver armazenando dados pode comprometer todo o trabalho.

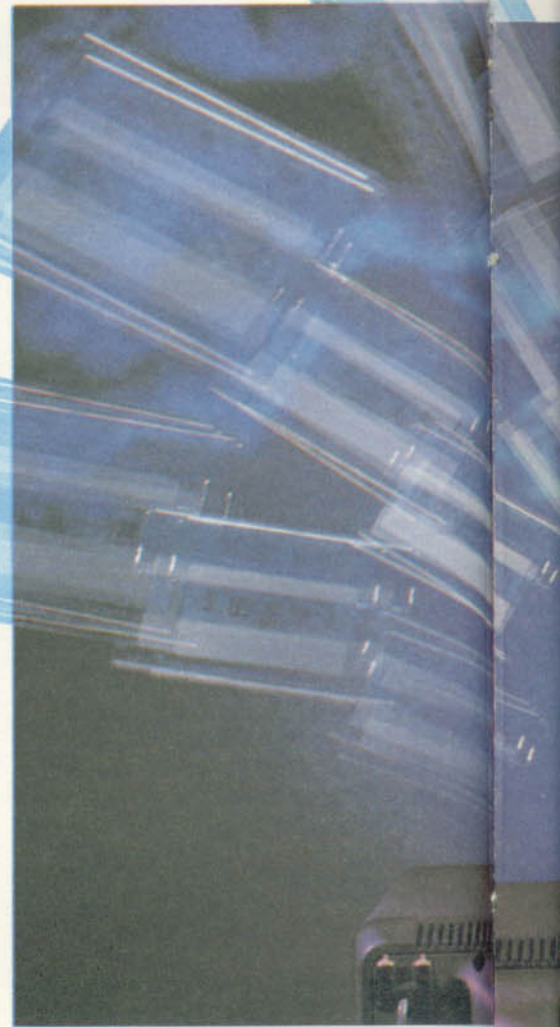
Se você tiver certeza de que quer abrir um novo arquivo, pressione S. Mas, se o arquivo já estiver armazenando dados e você não deseja apagá-los, pressione qualquer outra tecla, e o programa retornará ao menu principal.

O TAMANHO DOS CAMPOS

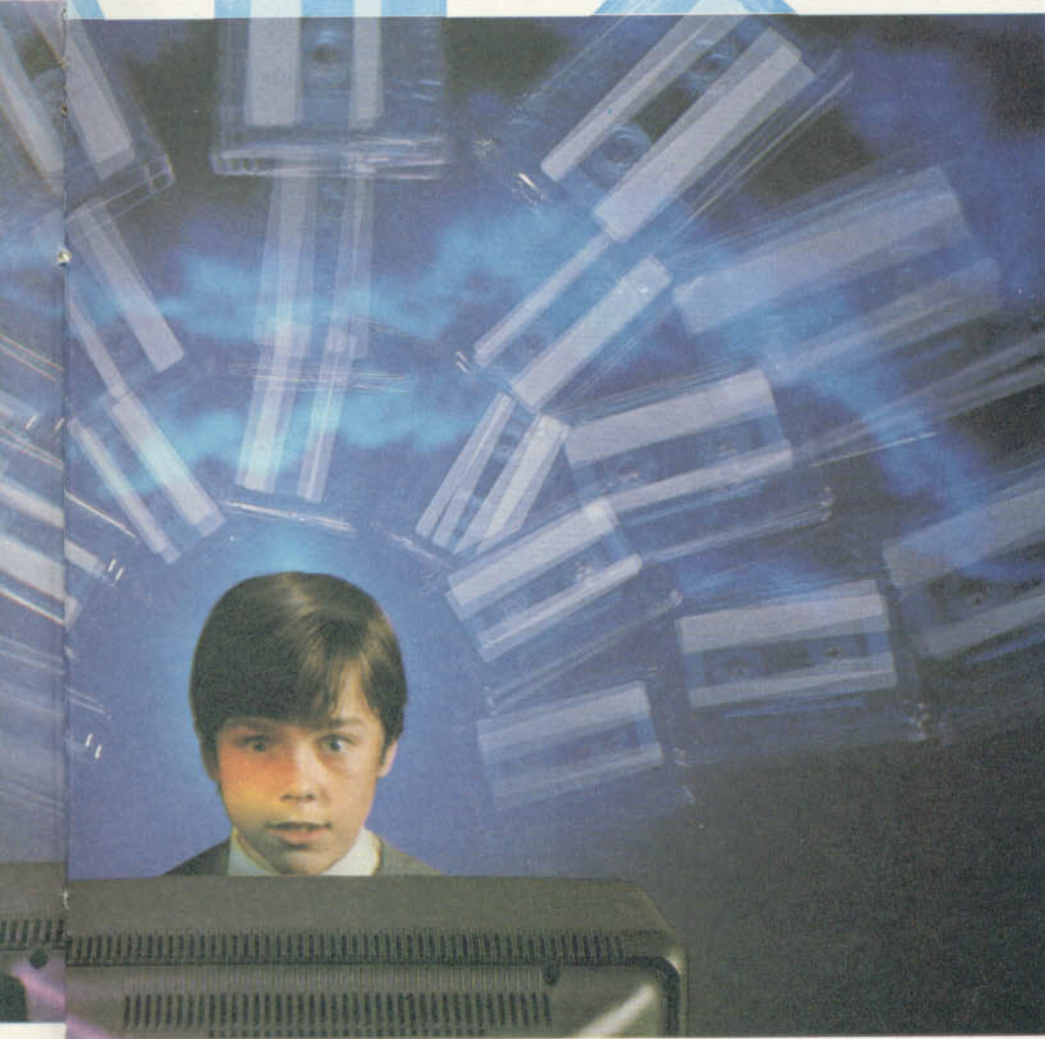
Uma vez que você tenha pressionado o S para continuar, o computador perguntará quantos campos você quer. "Campos" são os itens de informação que você quer armazenados em cada registro. Por exemplo, se você quer registrar dados sobre os seus amigos, poderiam ser definidos os seguintes campos: *nome*, *endereço*, *cidade* e *número do telefone* — quatro ao todo.

Este programa só permite um número máximo de oito campos para um registro individual; do contrário, ele não poderia exibi-los, todos, na tela ao mesmo tempo. Uma vez fornecido o número de campos a pergunta seguinte do computador será: "Qual o nome do campo 1?". (No exemplo acima, a resposta poderia ser NOME.)

A seguir, o programa pede a extensão do primeiro campo, isto é, o número máximo de caracteres que ele pode conter.

A extensão máxima para qualquer campo permitida no programa é de dezenove caracteres. Isso significa que, se a informação que se quer arquivar — um endereço, por exemplo — não couber nessa extensão, é preciso dividir o campo em duas ou mais partes. No caso de um endereço isso pode ser feito por meio da definição de campos separados para a rua, número da casa, cidade, código postal, etc.

De posse da informação sobre o primeiro campo, o computador fará a mesma pergunta sobre o segundo campo, o terceiro, e assim por diante. Obviamente, quanto menores forem os nomes dos campos e os números dos caracteres em cada campo, mais registros poderão ser retidos no arquivo.

Isto feito, o computador calculará para quantos registros ele tem espaço na memória. Esse número será exibido na tela. No programa para o TK-90X, você deve especificar quantos registros quer armazenar, evitando que o computador gaste tempo excessivo gravando ou lendo longos arquivos da fita.

- MANTENHA EM ORDEM AS SUAS COLEÇÕES
- UM PROGRAMA PRÁTICO DE ARQUIVAMENTO DISPONÍVEL
- UTILIZE EFICIENTEMENTE A MEMÓRIA DISPONÍVEL
- ESTABELEÇA UM NOVO ARQUIVO
- REVEJA SEU ARQUIVO
- COMO ARMAZENAR E RECUPERAR INFORMAÇÕES

COMO ENTRAR UM REGISTRO

Terminado o processo de criação de um novo arquivo, o programa trará de volta o menu principal, onde você deve selecionar a opção 2 (tecla 2), para iniciar a entrada da informação em seus registros.

No alto da tela, o computador manterá uma contagem dos registros que você já entrou, juntamente com o espaço total disponível na memória. Essa linha de informação dirá, por exemplo: "Você utilizou 10 dos 100 registros".

Logo abaixo disso, na tela, serão exibidos os nomes dos campos. O cursor aparecerá na última linha do vídeo, de tal modo que tudo que você escrever pelo teclado será registrado no campo exibido. Lembre-se de manter a informação digitada tão curta quanto possível e dentro da extensão máxima de caracteres que você estabeleceu para cada campo.

Quando você pressionar a tecla **<ENTER>** ou **<RETURN>**, a informação digitada será impressa próxima ao nome do campo. A linha inferior da tela será limpa e ficará pronta para o próximo campo. Esse procedimento começa com o primeiro campo no alto da tela e prossegue por toda a tela cada vez que você digita uma informação e pressiona **<ENTER>** ou **<RETURN>**. Quando você tiver entrado o último campo do registro, o computador irá para o próximo registro a ser entrado. Se você pressionar a tecla **<ENTER>** ou **<RETURN>** novamente, antes de começar a digitar o primeiro campo, o computador trará o menu principal de volta.

EXAMINE OS REGISTROS

Para examinar os registros que você entrou, selecione a opção número 3 do menu principal, "VER REGISTROS", pressionando a tecla 3. O primeiro registro será então exibido na tela — não necessariamente o primeiro que você introduziu, mas o primeiro de acordo com o próprio método de seleção do programa.

Os métodos de arranjo dos computadores variam muito pouco. Mas, de modo geral, eles selecionam os registros em ordem alfabética através do primeiro campo, o qual, em muitos casos, será "NOME". Para fazer isso, o computador examina a primeira letra do primeiro campo de todos os registros e os ordena alfabeticamente. Se mais de um registro tiver a mesma inicial, ele os ordena em função da segunda letra, e assim por diante.

Quando o primeiro campo contiver algarismos, o computador selecionará sempre um deles antes de qualquer letra; mas prosseguirá com o mesmo método de ordenação, dígito por dígito, quando estiver decidindo entre números, além de olhar para o número como um todo. Em outras palavras, se você colocar os números de 1 a 100 no primeiro campo, por exemplo, o computador ordenará em primeiro lugar os números 1, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19 e 100 antes de movimentar-se para o 2, 20, 21, etc. A maneira de se contornar esse problema é evitar o emprego de números no primeiro campo ou entrar os números com zeros à esquerda, como, por exemplo: 001, 002 ...010, 011 ... até 100.

Quando, por outro lado, se utiliza uma mistura de letras maiúsculas e mi-

núsculas, o computador escolhe primeiro as maiúsculas. Assim, "ABC Limitada" poderá ser antes de "Aarão e Companhia". Dependendo do que o arquivo de dados contiver, pode ser que o mais conveniente seja você digitar todas as informações apenas em letras maiúsculas, para resolver o problema. Se o seu computador aceitar caracteres acentuados, você poderá ter problemas na ordenação, pois na codificação brasileira para os caracteres (adotada, por exemplo, para os micros da linha MSX) as letras acentuadas aparecem depois das minúsculas, na ordem alfabética.

Quando você pedir para ver os registros, aparecerão as seguintes opções:

```
PROSSEGUE    RETORNA    MENU
```

escritas na parte inferior da tela. Se você pressionar a tecla P, o computador exibirá o registro seguinte; se você pressionar o R várias vezes, ele se movimentará rapidamente através de todo o arquivo, registro por registro.

Uma pressão na tecla R traz de volta à tela o registro anterior ao exibido. Assim, usando apenas o P e o R, você pode percorrer o arquivo, registro por registro, para a frente e para trás. Já a tecla M provoca o retorno ao menu principal, qualquer que seja o ponto onde você estiver no arquivo.

Embaixo da primeira linha de opções aparecerá uma segunda com as seguintes alternativas:

```
CORRIGE   APAGA   IMPRIME
```

Essas funções não constam do atual programa e serão explicadas no próximo artigo desta série. Portanto, nada acontecerá se você tentar usá-las na presente versão do programa.

ARMAZENE ARQUIVOS NA MEMÓRIA

À medida que você for entrando dados nos registros do seu arquivo, eles serão armazenados na memória principal (memória RAM) do computador, dentro do limite máximo no número de registros, calculado pelo programa. É necessário, portanto, armazenar esse arquivo em fita magnética, se você quiser desligar o computador; caso contrário, perderá toda a informação digitada. Posteriormente, quando você quiser modificar ou acrescentar dados, consultar ou listar o arquivo, poderá carregar de volta o arquivo para a memória RAM do computador, lendo-o em fita.

Na maioria das versões do programa listadas abaixo, o menu principal oferece as duas mencionadas opções ao usuário: gravar ou carregar o arquivo de dados. A única exceção é a versão para os micros da linha Sinclair, que gravam o programa juntamente com os dados contidos em memória.

Quando você quiser salvar o seu arquivo nos micros das linhas TRS-Color ou MSX, pressione a tecla 5 para selecionar a opção de armazenamento no menu principal, e o computador lhe pedirá que dê um nome qualquer ao arquivo. Uma vez que você tenha teclado o nome do arquivo e pressionado <**RETURN**> ou <**ENTER**>, o computador pedirá que você posicione o gravador para a gravação, apertando as teclas **RECORD** e **PLAY**, e pressionando = <**ENTER**> em seguida. Enquanto grava, o computador mantém você informado. O tempo de gravação depende do número de registros a ser armazenado.

Nos micros da linha Sinclair, a função de gravação é ativada de forma diferente da anterior. Neste caso, selecione primeiro a opção 7 (SAÍDA) no menu principal. Em seguida, usando o comando **SAVE** normal, digitado diretamente pelo teclado, acione o gravador e grave o programa, dando-lhe um nome adequado. Os dados serão gravados conjuntamente, conforme já foi explicado.

Posteriormente, quando quiser consultar o arquivo, você deve (nos micros das linhas citadas acima) carregar o computador em dois estágios. Inicialmente, é necessário carregar o programa, utilizando o comando normal **LOAD** (ou **CLOAD**) de sua máquina. Em seguida, deve-se mandar executar o programa, acionando o comando **RUN**, e selecionar a opção 6 no menu principal, "CARREGAR ARQUIVO". O computador pedirá então o nome do ar-

quivo desejado. Quando você tiver teclado o nome do arquivo e pressionado <**RETURN**> ou <**ENTER**>, a máquina ordenará: "POSICIONE O GRAVADOR E PRESSIONE <**ENTER**>". Feito isto, ele dirá novamente: "COLOQUE O GRAVADOR NA POSIÇÃO **PLAY** E PRESSIONE <**ENTER**>".

Assim que a fita cassete começar a rodar, o computador a examinará até encontrar o arquivo que você deseja, o qual será carregado (o que poderá demorar algum tempo). Concluída a operação, ele dirá que o arquivo foi carregado corretamente. Se o arquivo que você quiser não estiver na fita, o computador testará todos os arquivos que encontrar nela, até o final. De qualquer maneira, no final, o programa fará retornar o menu principal. Só então você poderá selecionar a opção 6 novamente e procurar outro arquivo para carregar.

Nos micros da linha Sinclair, como é o caso do TK-90X, os dados e o programa principal são carregados juntos. Uma vez que tiver carregado o primeiro arquivo utilizando o comando **LOAD** padrão, poderá carregar outros arquivos por meio da opção 6 no menu principal.

A versão do programa para computadores da linha Apple também tem funções de menu iguais às dos micros já considerados. A única diferença é que o programa utiliza arquivos em disquete, para armazenamento e recuperação de dados, pois não funciona satisfatoriamente com fita cassete.

Os programas listados contêm grandes intervalos em seus números de linhas — da linha 2000 à linha 6000.

Todavia, isso foi feito intencionalmente. Digite os programas exatamente como estão, pois as linhas que faltam serão utilizadas para incorporar as partes responsáveis pelas funções de modificação, apagamento, impressão e cruzamento de informações dos registros.

Os detalhes dessas opções estão no próximo artigo desta série.

```
20 PCLEAR1:CLEAR 11000:RS$="P R
MCAI":B$=CHR$(128)
30 CLS:PRINT@32,STRING$(8,B$);"
MENU";STRING$(3," ");"PRINCIPAL
";STRING$(8,B$)
40 PRINT @164,"1-ABRIR UM ARQUI
VO"
50 PRINT @196,"2-ENTRAR COM UM
REGISTRO"
60 PRINT @228,"3-VER REGISTROS"
70 PRINT @260,"4-OPCAO DE PROCU
RA"
80 PRINT @292,"5-SALVAR ARQUIVO
"
90 PRINT @324,"6-CARREGAR ARQUI
VO"
100 PRINT @356,"7-SAIDA
110 PRINT @481,"SELECIONE OPCAO
:";
120 IN$=INKEY$:IF IN$<"1"OR IN$
>"7" THEN 120
130 IF IN$ <>"1" AND IN$<>"6" A
ND R=0 AND IN$<>"7"THEN 120
140 SOUND 30,1:CLS:IN=VAL(IN$)
150 ON IN GOSUB 1000,2000,6000,
5000,7000,8000,9000
160 GOTO 30
1000 PRINT @41,"ABRIR UM ARQUIV
O":PRINT @231,"VOCE TEM CERTEZA
(S/N)?"
1010 IN$=INKEY$:IF IN$<>"S" AND
 IN$<>"N" THEN 1010
1020 IF IN$<>"S"THEN RETURN
1030 IF R>0 THEN RUN 9200
1040 CLS:PRINT @41,"ABRIR UM AR
QUIVO"
1050 PRINT @385,"NUMERO DE CAMP
OS(1-8)";:INPUT A: A=ABS(INT(A)
)
1060 IF A>8 OR A<1 THEN 1050
1070 DIM A(A),N$(A)
1080 PRINT @384,"":PRINT @96,""
:FOR N=1 TO A
1090 PRINT:PRINT"NOME DO CAMPO"
;N;"?";:LINEINPUT N$(N):N$(N)=L
EFT$(N$(N),10)
1100 PRINT"COMPRIMENTO DO CAMPO
";N;:INPUT A(N):A(N)=ABS(INT(A(
N)))
1110 IF A(N)>19 OR A(N)<1 THEN
1100
1120 TS=TS+A(N)
1130 NEXT:R=INT(11000/(5+5*A))-
1:PRINT" NUMERO MAXIMO DE REGIS
TROS=";R
1140 DIM A$(R,A):FOR I=1 TO 200
0:NEXT:RETURN
2000 G=0
2010 IF NR=R THEN 2180
2020 NR=NR+1
2030 CLS:PRINT @0,"VOCE USOU ";
NR-1;" DOS ";R;" REGISTROS DISP
ONIVEIS"
2040 FOR N=1 TO A:PRINT @32*N+3
2,N$(N);" :":PRINT @448,"":PRIN
T @416,""
2050 PRINT @416,"(ATE ";A(N);"C
ARACTERES)  ";:LINE INPUT A$(NR
,N)
2060 IF A$(NR,N)="" AND  N=1 TH
EN N=A:G=1:GOTO 2080
2070 A$(NR,N)=LEFT$(A$(NR,N),A(
N)):PRINT @32*N+45,A$(NR,N)
2080 NEXT
2090 IF G=1 THEN 2160
2100 C=NR:FOR F=1 TO 150:NEXT:I
F NR=1 THEN 2150
2110 IF A$(C,1)>=A$(C-1,1) THEN
 2150
2120 FOR N=1 TO A:X$=A$(C,N):A$
(C,N)=A$(C-1,N):A$(C-1,N)=X$:NE
XT:C=C-1
2130 IF C=1 THEN 2150
2140 GOTO 2110
2150 GOTO 2010
2160 NR=NR-1
2170 RETURN
2180 CLS3:PRINT @235," ARQUIVO
CHEIO ";:FOR G=1 TO 5: SCREEN 0
,1: FOR F=1 TO 500:NEXT
2190 SCREEN 0,0:FOR F=1 TO 500:
 NEXT F,G: RETURN
3000 RETURN:REM LINHA TEMPORARI
A
4000 RETURN:REM LINHA TEMPORARI
A
5000 RETURN:REM LINHA TEMPORARI
A
6000 D=1
6010 IF NR<1 THEN 6170
6020 GOSUB 8500
6030 PRINT @451,"pROSSEGUE    r
ETORNA   mENU  cORRIGE     aPAGA
     iMPRIME";
6040 IN$=INKEY$:IF IN$="" THEN
6040
6050 IN=INSTR(1,RS$,IN$)
6060 ON IN GOTO 6080,6080,6090,
6100,6110,6120,6130
6070 GOTO 6030
6080 D=D+1:GOTO 6140
6090 D=D-1:GOTO 6140
6100 RETURN
6110 GOSUB 3000:GOTO 6020
6120 GOSUB 4000:GOTO 6010
6130 GOSUB 10000:GOTO 6030
6140 IF D>NR THEN D=1
6150 IF D<1 THEN D=NR
6160 GOTO 6010
6170 CLS3:PRINT@233," ARQUIVO V
AZIO ";
6180 FOR G=1 TO 5:SCREEN 0,1:FO
```

COMO TORNAR UM PROGRAMA LONGO MAIS FÁCIL DE DIGITAR

Digitar um programa longo escrito por outra pessoa pode ser uma tarefa bastante trabalhosa e desanimadora, mesmo para os mais persistentes. Mas você pode torná-la mais fácil, e gratificante, digitando apenas uma curta seção de cada vez e testando-a em seguida.

Alguns programas estão estruturados em seções ou módulos mais ou menos independentes, tais como subrotinas, procedimentos e laços, mas se você não conseguir identificá-los digite apenas seções curtas de 20 ou 30 linhas.

Quando nada se sabe de programação BASIC, a chance de errar uma linha e não perceber é relativamente grande, pois a mudança de apenas um pequeno sinal ou letra já é suficiente para impedir que o programa funcione corretamente.

Eis aí um bom incentivo para você seguir também o curso de programação BASIC de INPUT, de modo sistemático e desde o começo.

```
R F=1 TO 300:NEXT:SCREEN 0,0:FO
R F=1 TO 300:NEXT F,G:RETURN
7000 AUDIOON:MOTORON:CLS:PRINT
@65,"POSICIONE O GRAVADOR E PRE
SSIONE <ENTER>";
7010 IN$=INKEY$:IF IN$<>CHR$(13
) THEN 7010
7020 MOTOROFF:PRINT @129,"COLOQ
UE O GRAVADOR EM POSICAO 'REC'
E PRESSIONE <ENTER>";
7030 IN$=INKEY$:IF IN$<>CHR$(13
) THEN 7030
7040 PRINT:INPUT" NOME DO ARQUI
VO ";FI$
7050 CLS6:PRINT @232,"GRAVANDO
";FI$;
7060 MOTORON:FOR I=1 TO 1000:NE
XT
7070 OPEN"O",#-1,FI$
7080 PRINT#-1,FI$,R,A,NR
7090 FOR N=1 TO A:PRINT#-1,N$(N
),A(N):NEXT
7100 C=1
7110 IF A$(C,1)="" THEN 7140
7120 FOR N=1 TO A:PRINT#-1,A$(C
,N):NEXT
7130 C=C+1:GOTO 7110
7140 CLOSE#-1:RETURN
8000 CLS:PRINT@70,"VOCE TEM CER
TEZA (S/N)?"
8010 IN$=INKEY$:IF IN$<>"S" AND
 IN$<>"N" THEN 8010
8020 IF IN$="N" THEN RETURN
8030 AUDIOON:MOTORON:CLS:PRINT
@65,"POSICIONE O GRAVADOR E PRE
SSIONE <ENTER>"
8040 IN$=INKEY$:IF IN$<>CHR$(13
) THEN 8040
8050 MOTOROFF:PRINT @129,"COLOQ
UE O GRAVADOR  NA POSICAO  'PLA
Y' E PRESSIONE <ENTER>"
8060 IN$=INKEY$:IF IN$<>CHR$(13
) THEN 8060
8070 IF R>0 THEN RUN 9210
8080 INPUT " NOME DO ARQUIVO";F
I$
8090 CLS7:PRINT @231,"PROCURAND
O ";
8100 OPEN "I",#-1,FI$
8110 INPUT #-1,FI$
8120 PRINT @231," ACHEI   ";FI$;
" ";
8130 INPUT#-1,R,A,NR
8140 DIM A(A),N$(A),A$(R,A)
8150 FOR N=1 TO A:INPUT#-1,N$(N
),A(N):NEXT
8160 C=1
8170 IFEOF(-1)THEN 8200
8180 FOR N=1 TO A:INPUT #-1,A$(
C,N)
8190 NEXT: C=C+1:GOTO 8170
8200 CLOSE #-1:RETURN
8500 CLS:PRINT@0,"NUMERO DO REG
ISTRO";D:FOR N=1 TO A:PRINT @32
*N+32,N$(N);" :";TAB(13);A$(D,N
):NEXT:RETURN
9000 CLS4 : PRINT @70,"VOCE TEM
 CERTEZA (S/N)?";
9010 IN$=INKEY$:IF IN$<>"S" AND
 IN$<>"N" THEN 9010
9020 IF IN$="N" THEN RETURN
9030 CLS:END
9200 GOSUB 1040:GOTO 9220
9210 GOSUB 8080
9220 B$=CHR$(128):RS$="P RMCAI"
:GOTO 30
10000 PRINT @451,"    CHEQUE IMP
RESSORA     cONT ";
10010 PRINT @480,"
                          ";
10020 IN$=INKEY$:IF IN$="" THEN
 10020
10030 IF IN$<>"C" THEN RETURN
10040 FOR Y=0 TO A+4:FOR X=0 TO
 31:P=PEEK(1024+X+Y*32):IFP>95
AND P<127 THEN P=P-64
10050 IF P>0 AND P<27 THEN P=P+
96
10060 IF P=0 THEN P=32
10070 PRINT #-2,CHR$(P);:NEXT:P
RINT#-2,CHR$(13);:NEXT
10080 FOR N=1 TO 3:PRINT#-2,CHR
$(13):NEXT
10090 RETURN
```

S

```
  5 LET R=0: LET U=0: LET V=1
 10 BORDER V: PAPER V: INK 7:
 POKE 23609,20: POKE 23658,8
 100 CLS : PRINT  INVERSE V;AT
V,10; " M E N U "
 110 PRINT AT 5,6;"1-Abrir um a
rquivo" ''TAB 6;"2-Entrar com
um registro"''TAB 6;"3-Ver reg
istros"''TAB 6;"4-Opcao de pro
cura"''TAB 6;"6-Carregar arqui
vo"''TAB 6;"7-Saida";#V;TAB 6;
"-SELECIONE OPCAO-"
 500 LET  I$=INKEY$ : IF  I$=""
 THEN  GOTO 500
 510 IF  I$<"1" OR I$>"7"  THEN
 GOTO 500
 520 IF R=U  AND  I$<>"1"  AND
I$ <>"6" AND I$<>"7" THEN
 GOTO 500
 530 SOUND .1,10: CLS : GOSUB (
 CODE I$ -48)*1000: GOTO 100
1000 PRINT AT 7,8;"Voce tem cer
teza?": PAUSE U: IF INKEY$="" T
HEN  GOTO 1000
1010 IF INKEY$<>"S" THEN  RETUR
N
1020 PRINT  INVERSE V;AT 10,5;"
 CRIAR UM NOVO ARQUIVO "
1030 INPUT AT 0,0;"Numero de ca
mpos(1-8)? ";A: IF a<1 OR a>8 T
HEN  GOTO 1030
1040 DIM A(A): DIM B(A+V): DIM
N$(A,10): LET T=U: FOR N=V TO A
1050 INPUT AT 0,0;"Nome do camp
o ";(N);" ? "; LINE N$(N)
1060 INPUT AT V,0;"Comprimento
do campo ";(N);" ? ";A(N): IF A
(N)>50 THEN  GOTO 1060
1070 LET B(N)=T: LET T=T+A(N):
NEXT N: LET B(N)=T
1080 PRINT AT 16,2;"Espaco para
 ";INT (((PEEK 23730+256*PEEK 2
3731)-29500)/T);" registros"
1090 INPUT "Quantos registros?
";R: DIM A$(R,T): RETURN
2000 LET C=V
2010 IF A$(C,V)=" " THEN  GOTO
2100
2020 IF C=R THEN  GOTO 2500
2030 LET C=C+V: GOTO 2010
2100 PRINT AT 0,0;"Voce usou ";
C-V;" dos ";R;" registros    d
isponiveis"
2110 FOR N=V TO A: PRINT  INVER
SE V;AT V+N*2,U;N$(N); INVERSE
0;AT V+N*2,12; FLASH V;"?": INP
UT "(ate ";(A(N));" caracteres)
", LINE A$(C,B(N)+V TO B(N+V)):
 PRINT AT V+2*N,12;A$(C,B(N)+V
TO B(N+V)): NEXT N
2120 FOR F=V TO 150: NEXT F: IF
 C=V THEN  RETURN
2130 LET N=C
2140 IF A$(C)>=A$(C-V) THEN  RE
TURN
2150 LET X$=A$(C): LET A$(C)=A$
(C-V): LET A$(C-V)=X$: LET C=C-
V: IF C=V THEN  RETURN
2160 GOTO 2140
2500 CLS : PRINT  FLASH 1;AT 10
,2;" A R Q U I V O   C H E I O
": FOR F=V TO 400: NEXT F: RETU
RN
3000 LET D=V: IF A$(V,V)=" " TH
EN  RETURN
3010 IF D=U THEN  LET D=V
3015 IF D-V=R THEN  LET D=D-V
3020 IF A$(D,V)=" " THEN  LET D
=D-V
3030 GOSUB 9500
3040 IF OP=V THEN  LET D=D+V: G
OTO 3010
3050 IF OP=2 THEN  LET D=D-V: G
OTO 3010
3060 IF OP=3 THEN  RETURN
3070 IF OP=4 THEN  GOSUB 8000
3080 IF OP=5 THEN  LET MD=V: GO
SUB 9000: IF D=U THEN  RETURN
3090 GOTO 3030
4000 RETURN : REM LINHA TEMPORA
RIA
5000 INPUT "Digite o nome do ar
quivo  "; LINE Q$: IF LEN Q$<V
OR LEN Q$>10 THEN  GOTO 5000
5010 SAVE Q$ LINE 10: RETURN
6000 PRINT AT 8,U;"Digite o nom
e do  arquivo  a sercarregado,
ou somente <ENTER> para carrega
r o primeiro arquivo"
6010 INPUT  LINE X$: IF LEN X$>
10 THEN  GOTO 6010
6020 PRINT AT 13,U;"PRESSIONE P
LAY NO GRAVADOR": LOAD X$
7000 PRINT AT 10,8;"Voce tem ce
rteza?": IF INKEY$="" THEN  GOT
O 7000
7010 IF INKEY$<>"S" THEN  RETUR
N
7020 RAND USR U
8000 RETURN : REM LINHA TEMPORA
RIA
9000 RETURN : REM LINHA TEMPORA
RIA
9500 PRINT AT U,U;"Registro num
ero ";D;"  ": FOR N=V TO A: PRI
NT  INVERSE V;AT V+2*N,U;N$(N);
 INVERSE U;TAB 12;A$(D,B(N)+V T
O B(N+V)): NEXT N
9510 PRINT  INVERSE V;AT 20,U;"
P(rossegue)  R(etorna)   M(enu)
  E(menda)    A(paga)  I(mprim
e)"
```

```
9520 IF INKEY$="" THEN  GOTO 95
20
9530 LET V$=INKEY$: IF V$="I" T
HEN  COPY : LPRINT : LPRINT : L
PRINT : GOTO 9520
9540 LET OP=U: IF V$="P" THEN
LET OP=V: LET MO=V
9550 IF V$="R" THEN  LET OP=2:
LET MO=-V
9560 IF V$="M" THEN  LET OP=3
9570 IF V$="E" THEN  LET OP=4
9580 IF V$="A" THEN  LET OP=5
9590 IF OP=U THEN  GOTO 9520
9600 SOUND .1,10: RETURN
```

```
10 KEYOFF:MOTOROFF
20 CLS:CLEAR:R$="P RMCAI"
30 CLStLOCATE 5,1:PRINT" M E N
U   P R I N C I P A L "
40 LOCATE 9,5:PRINT"1:-ABRIR UM
 ARQUIVO"
50 LOCATE 9,7:PRINT"2:-ENTRADA
DE REGISTROS"
60 LOCATE 9,9:PRINT"3:-LISTAR R
EGISTROS"
70 LOCATE 9,11:PRINT"4:-BUSCA D
E INFORMAÇÕES"
80 LOCATE 9,13:PRINT"5:-GRAVAR
UM ARQUIVO"
90 LOCATE 9,15:PRINT"6:-CARREGA
R UM ARQUIVO"
100 LOCATE 9,17:PRINT"7:-FIM DE
 PROGRAMA"
110 LOCATE 14,19:PRINT"OPÇÃO: "
;
120 IN$=INKEY$:IF IN$<"1"ORIN$>
"7"THEN120
130 IFIN$<>"1"ANDIN$<>"6"ANDIN$
<>"7"ANDR=0THEN 120
140 PRINTCHR$(7) :CLS:IN=VAL(IN
$)
150 ON IN GOSUB 1000,2000,6000,
5000,7000,8000,9000
160 GOTO 30
1000 LOCATE 7,0:PRINT"MONTAR UM
 NOVO ARQUIVO":LOCATE 7,14:PRIN
T"Você tem certeza (S/N)?";
1010 IN$=INKEY$:IFIN$<>"S"ANDIN
$<>"N"THEN 1010
1020 IFIN$<>"S"THEN RETURN
1030 IFR>0THEN RUN 9200
1040 CLS:LOCATE 7,0:PRINT"MONTA
R UM NOVO ARQUIVO"
1050 LOCATE 4,2:PRINT"Número de
 campos (1-8) ";:INPUT A:A=ABS(
INT(A))
1060 IF A>8 OR A<1 THEN1050
1070 DIM A(A),N$(A)
1080 LOCATE 0,6:FOR N=1TOA
1090 PRINT:PRINT"Nome do campo
";N;"? ";:LINEINPUTN$(N):N$(N)=
LEFT$(N$(N),10)
1100 PRINT"Tamanho do campo ";N
;:INPUTA(N):A(N)=ABS(INT(A(N)))
1110 IFA(N)>25ORA(N)<1THEN1100
1120 TS=TS+A(N)
1130 NEXT:R=INT(3000/(TS+3*A)):
PRINT:PRINT"Número máximo de re
gistros =>";R
1140 DIMA$(R+1,A):FORI=1TO1000:
NEXT:RETURN

2000 G=0
2010 IFNR=RTHEN2190
2020 NR=NR+1
2030 CLS:PRINTNR-1;" de ";R;" r
egistros em uso"
2040 FORN=1TOA:LOCATE0,N*2+2:PR
INTN$(N);": ":LOCATE0,22:PRINTC
HR$(5);:LOCATE0,23:PRINTCHR$(5)
;
2050 LOCATE0,23:PRINT"Até ";A(N
);" caracteres";
2060 LOCATE0,21:PRINTCHR$(21);:
LINEINPUTA$(NR,N)
2070 IFA$(NR,N)=""ANDN=1THENN=A
:G=1:GOTO2090
2080 A$(NR,N)=LEFT$(A$(NR,N),A(
N)):LOCATE12,N*2+2:PRINTA$(NR,N
)
2090 NEXT
2100 IFG=1THEN2170
2110 C=NR:FORF=1TO500:NEXT:IFNR
=1THEN2160
2120 IFA$(C,1)>=A$(C-1,1)THEN21
60
2130 FORN=1TOA:SWAPA$(C,N),A$(C
-1,N):NEXT:C=C-1
2140 IFC=1THEN2160
2150 GOTO2120
2160 GOTO2010
2170 NR=NR-1
2180 RETURN
2190 CLS:LOCATE10,15:PRINT"ARQU
IVO CHEIO!":FORF=1TO1000:NEXT
2200 RETURN
3000 RETURN:REM LINHA TEMPORARI
A
4000 RETURN:REM LINHA TEMPORARI
A
5000 RETURN:REM LINHA TEMPORARI
A
6000 D=1
```

```
6010 IFNR<1THEN6170
6020 GOSUB8500
6030 LOCATE3,22:PRINT"[P]rosseg
ue   [R]etorna   [M]enu   [C]o
rrige    [A]paga     [I]mprime"
;
6040 IN$=INKEY$:IFIN$=""THEN604
0
6050 IN=INSTR(1,R$,IN$)
6060 ON IN GOTO 6080,6080,6090,
6100,6110,6120,6130
6070 GOTO6030
6080 D=D+1:GOTO6140
6090 D=D-1:GOTO6140
6100 RETURN
6110 GOSUB3000:GOTO6020
6120 GOSUB4000:GOTO6010
6130 GOSUB10000:GOTO6020
6140 IFD>NRTHEND=1
6150 IFD<1THEND=NR
6160 GOTO 6010
6170 CLS:LOCATE10,15:PRINT"ARQU
IVO VAZIO!"
6180 FORF=1TO500:NEXT:RETURN
7000 MOTOR:CLS:LOCATE0,5:PRINT"
Posicione a fita e tecle [retur
n]"
7010 IN$=INKEY$:IFIN$<>CHR$(13)
THEN7010
7020 MOTOR:PRINT:PRINT:PRINT"Co
loque o gravador em modo de gra
vação e pressione [return]"
7030 IN$=INKEY$:IFIN$<>CHR$(13)
THEN7030
7040 PRINT:INPUT"Nome do arquiv
o ";FL$ :F$="CAS:"+FL$
7050 LOCATE10,22:PRINT"Gravando
...";
7060 MOTOR:FORI=1TO1000:NEXT
7070 OPEN FL$ FOR OUTPUT AS#1
7080 PRINT #1,FL$;",";R,A,NR
7090 FORN=1TOA:PRINT#1,N$(N);",
";A(N):NEXT
7100 C=1
7110 IFA$(C,1)=""THEN7140
7120 FORN=1TOA:PRINT#1,A$(C,N):
NEXT
7130 C=C+1:GOTO7110
7140 CLOSE#1:RETURN
8000 CLS:LOCATE10,15:PRINT"Você
 tem certeza? (S/N) "
8010 IN$=INKEY$:IFIN$<>"S"ANDIN
$<>"N"THEN8010
8020 IFIN$="N"THENRETURN
8030 MOTOR:CLS:LOCATE0,3:PRINT"
Posicione a fita e tecle [retur
n]"
8040 IN$=INKEY$:IFIN$<>CHR$(13)
THEN8040
8050 MOTOR:LOCATE0,7:PRINT"Colo
que o gravador em modo de repro
dução e tecle [return]"
8060 IN$=INKEY$:IFIN$<>CHR$(13)
THEN8060
8070 IFR>0THENRUN9210
8080 PRINT:INPUT"Nome do arquiv
o ";FL$:F$="CAS:"+FL$
8090 LOCATE10,22:PRINT"Procuran
do..."
8100 OPEN F$ FOR INPUT AS#1
8110 INPUT #1,FL$
8120 CLS:LOCATE 10,15:PRINT"Ach
ei ";FL$
8130 INPUT#1,R,A,NR
8140 DIMA(A),N$(A),A$(R,A)
8150 FORN=1TOA:INPUT#1,N$(N),A(
N):NEXT
8160 C=1
8170 IFEOF(1)THEN8200
8180 FORN=1TOA:INPUT#1,A$(C,N)
8190 NEXT:C=C+1:GOTO8170
8200 CLOSE#1:RETURN
8500 CLS:PRINT"Número do regist
ro =>";D:FORN=1TOA:LOCATE0,N*2+
2:PRINTN$(N);" :";:LOCATE13:PRI
NTA$(D,N):NEXT:RETURN
9000 CLS:LOCATE10,15:PRINT"Você
 tem certeza? (S/N) "
9010 IN$=INKEY$:IFIN$<>"S"ANDIN
$<>"N"THEN9010
9020 IFN$="N" THENRETURN
9030 CLS:END
9200 GOSUB1040:GOTO9220
9210 GOSUB8080
9220 R$="P RMCAI"
10000 CLS:LOCATE10,15:PRINT"CHE
QUE A IMPRESORA"
10010 PRINT:LOCATE12:PRINT"ALIN
HE O PAPEL"
10020 IN$=INKEY$:IFINKE$=""THEN
10020
10040 LPRINT:LPRINT"Registro nú
mero ";D
10050 FORX=1TOA:LPRINT:LPRINT N
$(X);" :";A$(D,X):NEXT
10060 LPRINT:LPRINT:LPRINT:LPRI
NT
10070 CLS:RETURN
```

```
20  TEXT : CLEAR :D$ =  CHR$ (4
)
25  ONERR  GOTO 11000
30  HOME : VTAB 1: HTAB 6: INVE
RSE : PRINT " M E N U    P R I
N C I P A L ": NORMAL
40  VTAB 5: HTAB 10: PRINT "1:-
ABRIR UM ARQUIVO"
50  PRINT : HTAB 10: PRINT "2:-
ENTRADA DE REGISTROS"
60  PRINT : HTAB 10: PRINT "3:-
LISTAR REGISTROS"
70  PRINT : HTAB 10: PRINT "4:-
BUSCA DE INFORMACOES"
80  PRINT : HTAB 10: PRINT "5:-
GRAVAR O ARQUIVO"
90  PRINT : HTAB 10: PRINT "6:-
CARREGAR UM ARQUIVO"
100  PRINT : HTAB 10: PRINT "7:
-FIM DE PROGRAMA"
110  VTAB 20: HTAB 15: PRINT "O
PCAO: ";
120  GET IN$
130  IF IN$ < "1" OR IN$ > "7"
THEN 110
135  IF R = 0 AND IN$ <  > "1"
AND IN$ <  > "6" AND IN$ <  > "
7" THEN 110
140  PRINT  CHR$ (7): HOME :IN
=  VAL (IN$)
150  ON IN GOSUB 1000,2000,6000
,5000,7000,8000,6520
160  GOTO 30
1000  HTAB 8: INVERSE : PRINT "
 MONTAR UM NOVO ARQUIVO ": VTAB
 15: HTAB 10: PRINT " VOCE CONF
IRMA? (S/N) ";
1005  NORMAL
1010  GET IN$: IF IN$ <  > "S"
AND IN$ <  > "N" THEN 1010
1020  IF IN$ <  > "S" THEN  RET
URN
1030  IF R > 0 THEN  CLEAR :D$
=  CHR$ (4):IN = 1: HOME : GOTO
 150
1040  HOME : HTAB 8: INVERSE :
PRINT " MONTAR UM NOVO ARQUIVO
": NORMAL
1050  VTAB 3: HTAB 5: PRINT "NU
MERO DE CAMPOS (1-8): ";: INPUT
 A
1060  IF A > 8 OR A < 1 THEN 10
50
1070  DIM A(A),N$(A)
1080  VTAB 7: FOR N = 1 TO A
1090  PRINT "NOME DO CAMPO ";N;
" ";: INPUT N$(N):N$(N) =  LEFT
$ (N$(N),10)
1100  PRINT "TAMANHO DO CAMPO "
;N;" ";: INPUT A(N):A(N) =  ABS
 ( INT (A(N)))
1110  IF A(N) < 1 OR A(N) > 25
THEN 1100
1120 TS = TS + A(N): PRINT
1130  NEXT :R =  INT (( FRE (0)
 - 5000) / (TS + 2 * A)): IF R
> 4000 THEN R = 4000
1135 R = 3: REM  RETIRAR ESTA L
INHA!!!!!!!!!!!!!!!************
************
1140  PRINT : PRINT "NUMERO MAX
IMO DE REGISTROS: ";R
1150  DIM A$(R,A): FOR I = 1 TO
 2000: NEXT : RETURN
2000 G = 0
2010  IF NR = R THEN 2180: REM
 CHECK GOTO
2020 NR = NR + 1
2030  HOME : PRINT NR - 1;" DE
";R;" REGISTROS EM USO"
2040  FOR N = 1 TO A: VTAB N *
2 + 2: PRINT N$(N);":": VTAB 23
: CALL  - 958
2050  VTAB 24: PRINT "ATE ";A(N
);" CARACTERES";: VTAB 23: HTAB
 1: INPUT A$(NR,N)
2060  IF A$(NR,N) = "" AND N =
1 THEN NR = NR - 1: RETURN
2070 A$(NR,N) =  LEFT$ (A$(NR,N
),A(N)): VTAB N * 2 + 2: HTAB 1
3: PRINT A$(NR,N)
2080  NEXT
2090  IF A$(NR,1) >  = A$(NR -
1,1) THEN 2150
2100  FOR C = NR TO 2 STEP  - 1
2110  FOR N = 1 TO A:X$ = A$(C,
N):A$(C,N) = A$(C - 1,N):A$(C -
 1,N) = X$: NEXT
2120  NEXT
2150  FOR F = 1 TO 1000: NEXT :
 GOTO 2010
2180  VTAB 3: HTAB 1: CALL  - 9
58: VTAB 15: HTAB 10: FLASH : P
RINT " ARQUIVO CHEIO! "
2190  PRINT  CHR$ (7): FOR F =
1 TO 1000: NEXT : PRINT  CHR$ (
7): RETURN
3000  RETURN : REM  LINHA TEMPO
```

```
RARIA
4000  RETURN : REM  LINHA TEMPO
RARIA
5000  RETURN : REM  LINHA TEMPO
RARIA
6000  IF NR < 1 THEN  VTAB 15:
HTAB 12: FLASH : PRINT " ARQUIV
O VAZIO! ": FOR F = 1 TO 1000:
NEXT : RETURN
6010 D = 1
6020  POKE 35,22: VTAB 23: INVE
RSE : HTAB 4: PRINT "P";: HTAB
18: PRINT "R";: HTAB 32: PRINT
"M"
6030  VTAB 24: HTAB 8: PRINT "C
";: HTAB 20: PRINT "A";: HTAB 3
0: PRINT "I";: NORMAL
6040  VTAB 23: HTAB 5: PRINT "R
OSSEGUE";: HTAB 19: PRINT "ETOR
NA";: HTAB 33: PRINT "ENU"
6050  VTAB 24: HTAB 9: PRINT "O
RRIGE";: HTAB 21: PRINT "PAGA";
: HTAB 31: PRINT "MPRIME";
6060  GOSUB 6500
6070  GET IN$
6080  IF IN$ =  CHR$ (32) OR IN
$ = "P" THEN D = D + 1: IF D >
NR THEN D = 1
6090  IF IN$ = "R" THEN D = D -
 1: IF D < 1 THEN D = NR
6100  IF IN$ <  > "I" AND IN$ <
  > "A" AND IN$ <  > "C" AND IN
$ <  > "M" THEN 6060
6110  POKE 35,24
6115  IF IN$ = "M" THEN  RETURN

6120  IF IN$ = "I" THEN  GOSUB
10000: GOTO 6020
6130  IF IN$ = "A" THEN  GOSUB
4000: GOTO 6020
6140  IF IN$ = "C" THEN  GOSUB
3000: GOTO 6020
6150  GOTO 6060
6500  VTAB 1: HTAB 1: CALL  - 9
58: PRINT "NUMERO DO REGISTRO =
>";: INVERSE : PRINT D: NORMAL
6510  FOR I = 1 TO A: VTAB I *
2 + 2: PRINT N$(I);: HTAB 13: P
RINT A$(D,I): NEXT : RETURN
6520  END
7000  HOME : VTAB 10: HTAB 10:
PRINT "GRAVAR O ARQUIVO"
7005  PRINT : PRINT : HTAB 5: I
NPUT "NOME DO ARQUIVO =>";AR$
7010  PRINT : HTAB 5: PRINT "CO
NFERE ? ";: GET IN$
7015  IF IN$ <  > "S" THEN  RET
URN
7020  PRINT : PRINT D$;"OPEN";A
R$: PRINT D$;"DELETE";AR$
7030  PRINT D$;"OPEN";AR$
7040  PRINT D$;"WRITE";AR$
7050  PRINT R: PRINT A: PRINT N
R
7055  FOR I = 1 TO A: PRINT N$(
I): PRINT A(I): NEXT
7060  FOR J = 1 TO NR
7070  FOR I = 1 TO A
7080  PRINT A$(J,I): NEXT : NEX
T
7090  PRINT D$;"CLOSE";AR$
7100  RETURN
8000  CLEAR :D$ =  CHR$ (4)
8010  VTAB 10: HTAB 10: PRINT "
CARREGAR DADOS"
8020  PRINT : HTAB 5: INPUT "NO
ME DO ARQUIVO =>";AR$
8025  HTAB 5: PRINT "CONFERE ?
";: GET IN$
8027  IF IN$ <  > "S" THEN  GOT
O 30
8030  PRINT D$;"OPEN";AR$
8040  PRINT D$;"READ";AR$
8050  INPUT R,A,NR: DIM A$(R,A)
,N$(A),N(A)
8060  FOR I = 1 TO A: INPUT N$(
I),A(I): NEXT
8066  FOR J = 1 TO NR
8069  FOR I = 1 TO A
8071  INPUT A$(J,I): NEXT : NEX
T
8073  PRINT D$;"CLOSE";AR$
8078  GOTO 30
10000  HOME : VTAB 23: HTAB 9:
PRINT " CHEQUE A IMPRESSORA! "
10010  HTAB 12: PRINT " ALINHE
O PAPEL ";
10015  GET IN$
10020  VTAB 1: PRINT : PRINT  C
HR$ (4);"PR#1"
10030  PRINT : PRINT "REGISTRO
NUMERO ";D
10040  FOR X = 1 TO A: PRINT :
PRINT N$(X);" :"; TAB( 15);A$(D
,X): NEXT
10050  PRINT : PRINT : PRINT :
PRINT : PRINT
10055  PRINT  CHR$ (4);"PR#0"
10060  HOME : RETURN
11000  PRINT D$;"CLOSE": GOTO 2
0
```

O QUE É UMA BASE DE DADOS?

O programa de arquivamento de dados listado neste artigo e no próximo pode ser utilizado em muitos tipos de aplicações, pois o usuário tem a oportunidade de definir, ele mesmo, os **campos** *ou diferentes itens de informação que constituirão a ficha de cadastramento a ser usada.*

Em computação, esse tipo de sistema flexível, ou programável pelo usuário final, é comumente chamado de **sistema gerenciador de base de dados***: além de possuir as funções essenciais de qualquer sistema de cadastramento, esse sistema também tem um módulo de criação do* **lay out** *do arquivo, conforme o uso que ele terá.*

Uma base de dados, na definição mais estrita do termo, é um conjunto de arquivos de computador, que tem a mesma finalidade do ponto de vista do armazenamento e recuperação de informações. Normalmente, portanto, uma dessas bases inclui mais de um arquivo: por exemplo, em uma base de dados bibliográficos (artigos de revistas), teríamos o arquivo-mestre, ou arquivo-tombo, o arquivo de índice por autores, o arquivo de índice por assuntos, o arquivo de nomes de revistas, e assim por diante. Todos eles podem ser inter-relacionados por meio de um ou mais campos em comum (por exemplo, o número de tombo do artigo, e o código da revista).

Entretanto, podemos considerar, de forma simplificada, que uma base de dados com apenas um arquivo também pode cair nessa definição. É o caso do programa listado em BASIC neste artigo.

A única condição essencial para que ele continue sendo chamado de um sistema gerenciador de base de dados é que o usuário possa definir livremente os campos de informação que quer ter em cada ficha (desde que, é claro, sejam respeitados os limites intrínsecos do programa).

MODELOS DE BASES DE DADOS

Existem diversas formas de organização da informação em uma base de dados. Normalmente, todas as formas têm pelo menos uma característica em comum: a subdivisão do arquivo em registros individuais.

A forma pela qual esses registros (ou os campos do registro) são organizados é que difere de caso para caso. Na verdade, a organização de uma base de dados é ditada pelo programa aplicativo que a controla.

A forma de organização mais comum para uma base de dados é a chamada **estrutura relacional**. *Ela é bastante fácil de entender e muito intuitiva, pois se equivale a uma tabela, ou planilha de dupla entrada, com linhas e colunas. Essa tabelc é chamada de* **relação**, *daí o nome da estrutura. Como você já deve ter percebido, o programa listado neste artigo é uma base de dados relacional.*

Na estrutura relacional, as linhas da tabela correspondem sempre aos registros (por exemplo, cada livro de uma biblioteca), ao passo que as colunas correspondem aos diversos campos (tais como o nome do autor, o título da revista, o ano de publicação, etc.).

AS PLACAS DE SINALIZAÇÃO

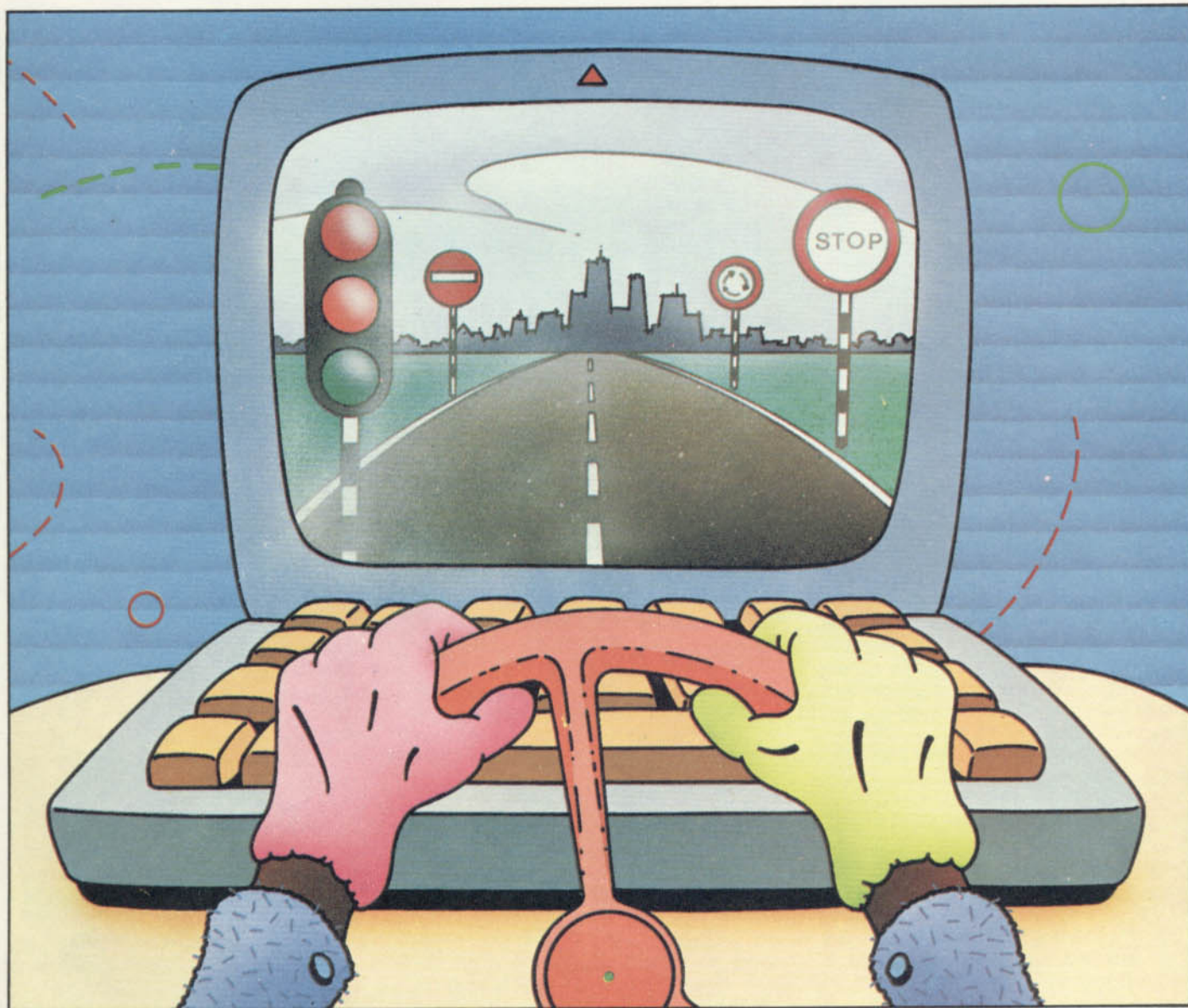
Universalmente adotados, os sinais de trânsito servem para orientar o fluxo do tráfego. No mundo dos micros, os comandos GOTO e GOSUB cumprem uma função semelhante.

Uma das instruções mais importantes em programação BASIC é o **GOTO** ("vá para", em inglês). Sua função é alterar o fluxo de execução de um programa, de modo que, ao invés de simplesmente executar as linhas do programa em ordem linear, o computador salta para a linha indicada na declaração **GOTO**.

Embora possa aparecer na tela como duas palavras separadas, **GOTO** normalmente é digitada como uma só expressão. Nos micros da linha Sinclair, você pressiona a tecla marcada **GOTO** para entrar a instrução no computador. Em outros tipos de micros, você digita **GOTO** sem o espaço entre **GO** e **TO**.

A palavra **GOTO** deve ser sempre seguida pelo número da linha para onde você quer que o programa salte, como no exemplo abaixo:

```
30 GOTO 125
```

Em alguns computadores, como os da linha Sinclair, em vez de um número, podemos ter uma expressão ou equação de cálculo (inclusive o nome de uma variável), que assume um valor numérico quando o programa é rodado.

- **GOTO** E **GOSUB** NA PRÁTICA
- PROGRAMAS PARA CÁLCULO, ADIVINHAÇÃO DE NOMES E JOGOS DE DADOS
- QUANDO E COMO CRIAR DESVIOS EM UM PROGRAMA
- UTILIZAÇÃO EM DESVIOS COMPLEXOS
- COMO ACELERAR AS SUB-ROTINAS

```
10 LET A=10
20 GOTO A+5  equivale a  GOTO 15
```

As declarações **GOTO** permitem também que você salte para trás em um programa, criando um laço. Esse tipo de laço é semelhante àquele que aprendemos a montar com as declarações **FOR...NEXT**, embora não haja, neste caso, um limite para o número de vezes que o programa o executa. O programa abaixo calcula o valor da hipotenusa de um triângulo retângulo (note de que maneira os valores em A e B são elevados ao quadrado: multiplicando-os por si mesmo, o que é mais rápido do que usar a operação de exponenciação).

```
10 PRINT "COMPRIMENTO DOS LADOS
 A,B EM CM"
15 INPUT A,B
20 LET C=SQR(A*A+B*B)
30 PRINT"O COMPRIMENTO DO LADO
C E";C;"CM"
40 GOTO 10
```

Na linha 20, a função **SQR** é utilizada para extrair a raiz quadrada (**SQR** é a abreviatura de *square root*, expressão inglesa que quer dizer exatamente "raiz quadrada") do que está entre parênteses, ou seja, a expressão toda significa: "extraia a raiz quadrada de A ao quadrado mais B ao quadrado, e armazene o resultado em C". (Esta é a fórmula de Pitágoras, $\sqrt{A^2 + B^2} = C^2$, para calcular o comprimento C da hipotenusa de um triângulo retângulo, cujos catetos são A e B.) A linha 30 imprime o valor resultante em C.

Em virtude da linha 40, o programa se auto-repetirá, pois cada vez que ele a alcança a declaração **GOTO** manda-o de volta à linha 10, e o programa é executado outra vez.

A única maneira de sair desse ciclo é interromper externamente o programa, através da tecla **<BREAK>** ou **<CTRL> <C>**, **<CTRL> <STOP>**, etc., dependendo do computador, ou então desligá-lo e começar de novo (pois o programa na memória se perde).

SALTOS PARA A FRENTE

A declaração **GOTO** pode ser utilizada para saltar para a frente, por sobre um bloco do programa. É o que acontece no pro grama a seguir, que simula um jogo de cara ou coroa:

```
5 FOR F=1 TO 500:NEXT
F:CLS
10 PRINT "ESTOU JOGANDO
A MOEDA ..."
20 FOR J=1 TO 3
30 FOR F=1 TO 250:NEXT F
40 PRINT".";
50 NEXT J
60 PRINT
70 IF RND(0)<.5 THEN GOTO 100
80 PRINT"E DEU COROA!"
90 GOTO 5
100 PRINT"E DEU CARA!"
110 GOTO 5
```

Digite o programa abaixo usando somente maiúsculas, se o seu computador for compatível com o ZX-81:

```
 5  PAUSE 50: CLS
10 PRINT "Estou jogando a moe
da...";
20 FOR j=1 TO 3
```

```
 30 PAUSE 25
 40 PRINT ".";
 50 NEXT j
 60 PRINT
 70 IF RND<.5 THEN  GOTO 100
 80 PRINT "E' coroa!!!"
 90 GOTO 5
100 PRINT "E' cara!!!"
110 GOTO 5
```

```
5 FOR F=1 TO 500 : NEXT
6 CLS
10 PRINT "ESTOU JOGANDO A
MOEDA..."
20 FOR J=1 TO 3
30 FOR F=1 TO 250 : NEXT F
40 PRINT ".";
50 NEXT J
60 PRINT
70 IF RND(1)<.5 THEN GOTO 100
80 PRINT "E DEU COROA !"
90 GOTO 5
100 PRINT "E DEU CARA !"
110 GOTO 5
```

A função **RND** na linha 70 sorteia um número aleatório entre 0 e 1. Aqui, a declaração **GOTO** faz parte da declaração **IF**. Se o número selecionado pelo computador for menor do que 5, o computador saltará para a frente, para a linha 100. Se ele não satisfizer essa condição, o computador executará normalmente a linha seguinte do programa — a linha 80 —, e quaisquer outros comandos porventura existentes na linha 70 serão desprezados.

Essa condição significa que a linha 70 constrói uma bifurcação no programa. Existem, neste caso, duas alternativas: ou o computador executa as linhas 70, 80 e 90, para exibir na tela a mensagem "E DEU CARA?", ou, então, executa as linhas 70, 100 e 110, para exibir "E DEU COROA!". Isso é feito repetidas vezes, de forma totalmente aleatória, como uma espécie de lançamento rápido de uma "moeda eletrônica".

As linhas 90 e 100 também contêm declarações **GOTO**. Qualquer que seja o desvio tomado pelo programa, elas o mandarão de volta ao início, na linha 5, para começar outro sorteio.

Mais uma vez, você notará que este programa não termina nunca. A única maneira de pará-lo é pressionar a tecla de interrupção do seu computador, ou desligá-lo.

DESVIOS MAIS COMPLEXOS

Nos computadores da linha Sinclair, a declaração **GOTO** não precisa ser seguida de uma constante numérica. Uma variável servirá: **GOTO A**, por exemplo, ou **GOTO(100 + INT(RND*6))**. Isso significa que a declaração **GOTO** pode proporcionar um tipo de desvio mais complexo ao seu programa; veja a seguir.

Digite o programa abaixo usando somente maiúsculas, se o seu computador for compatível com o ZX-81.

```
100 PRINT "Ola, qual e o seu n
ome?"
110 INPUT a$
120 GOTO (130+INT (RND*4)*10)
130 PRINT "E' um bonito nome,
";a$: GOTO 170
140 PRINT "E' um nome gozado,
";a$: GOTO 170
150 PRINT "Prazer em conhece-l
o, ";a$: GOTO 170
160 PRINT "Ola, ";a$;", eu sou
 seu computador."
170 STOP
```

A declaração **GOTO** na linha 120 dá um salto aleatório para a frente, em direção a qualquer uma das quatro linhas com as alternativas do programa, em função do resultado da expressão numérica. O número 130 na expressão pode ser somado a 0, 10, 20 ou 30, dependendo do resultado da expressão com RND. Assim, o programa poderá saltar para as linhas 130, 140, 150 ou 160.

O **GOTO 170**, após cada alternativa, faz com que o programa salte para a frente, ignorando as linhas intermediárias. Note que não necessitamos de um **GOTO 170** após a linha 160, pois o programa vai para a linha 170 de qualquer maneira (é a última alternativa de **GOTO** na linha 120).

Assim, este programa roda apenas uma vez, pois todos os **GOTO** instruem o computador para saltar para a instrução **STOP** na linha 170, sem formar laços de retorno.

A INSTRUÇÃO ON...GOTO

Os computadores da linha TRS, MSX e Apple têm um equivalente ao **GOTO** variável existente apenas nos computadores da linha Sinclair. É a declaração **ON...GOTO**. Esta última é bastante poderosa e útil em programação, e tem a seguinte forma (um exemplo):

```
ON A GOTO 100,200,300,400
```

Neste exemplo, quando a variável A for igual a 1, o programa saltará para o primeiro número de linha na lista que se segue à palavra **GOTO**, ou seja, a linha de número 100. Se A for igual a 2, o programa irá para a linha 200, e assim por diante. A lista de linhas após **GOTO** pode ter qualquer número de elementos, com um mínimo de 2, e um máximo determinado pelo número máximo de caracteres por linha de programa que o seu computador aceita (normalmente 254 ou 255 caracteres).

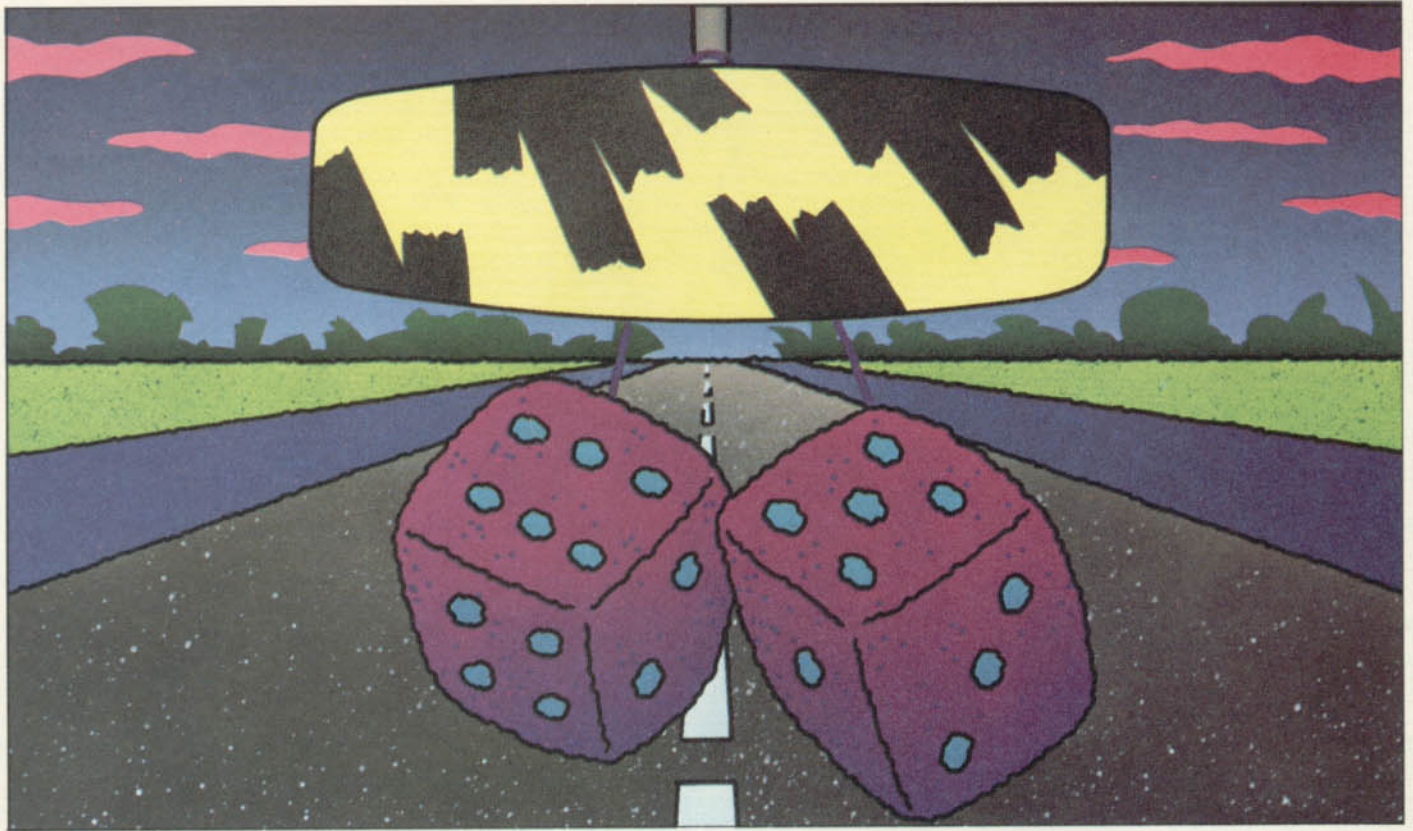
Novamente, esta declaração permite a programação de desvios complexos. O programa anterior passa a ser assim.

```
100 PRINT"OLA, QUAL E O SEU NOM
E?"
110 INPUT A$
120 ON RND(4) GOTO 130,140,150,
160
130 PRINT"E UM BONITO NOME, ";A
$:GOTO 170
140 PRINT"E UM NOME GOZADO, ";A
$:GOTO 170
150 PRINT"PRAZER EM CONHECE-LO,
 ";A$:GOTO 170
160 PRINT "OLA, ";A$;", EU SOU
O SEU COMPUTADOR."
170 END
```

```
100 PRINT "OLA, QUAL E' O SEU
NOME ?"
110 INPUT A$
120 ON INT(RND(1)*4)+1 GOTO
130, 140, 150, 160
130 PRINT "E' UM BONITO NOME, "
;A$:GOTO 170
140 PRINT "E' NOME GOZADO, "
;A$:GOTO 170
150 PRINT "PRAZER EM CONHECE-
LO, " ;A$:GOTO 170
160 PRINT "OLA, ";A$;", EU SOU
O SEU COMPUTADOR"
170 END
```

EVITE MÁS TÉCNICAS DE PROGRAMAÇÃO

O excessivo emprego de **GOTO** em um programa é considerado um mau estilo de programação. Uma razão para isso é que, mesmo em programas simples, uma declaração **GOTO** que o manda de volta para uma linha precedente pode criar um laço sem fim. Ora, esse laço só poderá ser interrompido por meio do teclado, pressionando-se, para isso, as teclas **<BREAK>**, **<STOP>**, **<CTRL>** **<C>**, etc. — ou desligando-se o computador. Isso é inconveniente, principalmente para o usuário do programa.

Mas a principal razão é que, permitindo que você salte, para trás ou para a frente, para qualquer ponto do programa, ao inteiro sabor de sua vontade, o uso do **GOTO** tende a quebrar a estrutura lógica do programa. Isso pode não parecer muito importante, se você estiver lidando com programas de 5 ou 10 linhas, mas pode ser vital, se você estiver escrevendo programas de 100 ou 1000 linhas.

Um bom estilo de programação requer que os programas sejam desenvolvidos em módulos lógicos, cada um desempenhando uma tarefa bem separada. Isso ajuda muito quando você tem que localizar erros de programação que ocorrem aleatoriamente, no momento em que o programa é rodado. Esse estilo de programação (conhecido como estruturado) o ajudará também a fazer programas mais simples, que poderão ser facilmente modificados no futuro.

COMO UTILIZAR O GOSUB

Existe outra ferramenta de programação que pode ser utilizada para substituir em grande parte o **GOTO**, quando este é problemático: é o **GOSUB**. Essa instrução é digitada com uma palavra e deve ser seguida por um número de linha (ou ainda por uma expressão numérica, em micros da linha Sinclair).

O **GOSUB** envia o computador para uma sub-rotina, que se inicia na linha especificada por ele. Uma sub-rotina é um conjunto de operações dentro de um programa que podem ser colocadas à parte, em uma espécie de "tijolo" lógico. Ela é freqüentemente utilizada quando uma operação deve ser repetida várias vezes ao longo de um programa. Assim, ao invés de escrever esta rotina toda vez que ela ocorre no programa, podemos colocá-la em um ponto qualquer, e simplesmente direcionar o computador para ela, quando necessário, e quantas vezes desejarmos.

A diferença decisiva entre um **GOTO** e um **GOSUB** é que no final de uma sub-rotina deve constar a palavra **RETURN** ("retornar", em inglês). Para entrar esta palavra, nos computadores da linha Sinclair deve-se simplesmente pressionar a tecla marcada com **RETURN**. Nos outros micros, digita-se a palavra por extenso, pressionando-se <**ENTER**> ou <**RETURN**> depois (não confunda a tecla de retorno de carro, ou transmissão, que é sempre indicada no texto de forma diferente, com a instrução **RETURN**).

A declaração **RETURN** envia o fluxo do programa de volta à linha ou ao comando que se segue ao **GOSUB** que solicitou ("chamou", na gíria dos programadores) a sub-rotina. Assim, o computador sempre "se lembra" de que ponto do programa foi efetuado o desvio para a sub-rotina e sempre volta a ele, para continuar o fluxo normal do programa, depois que a sub-rotina é completada. Desse modo, podemos colocar chamadas à mesma sub-rotina em qualquer ponto do programa.

No programa seguinte exemplificamos o uso do **GOSUB**. Ele simula um jogo de dados muito simples, uma espécie de "crepe". No jogo, um par de dados é arremessado, e o total dos pontos obtidos é anotado. Se os totais de dois arremessos sucessivos forem iguais, o jogo terminará. Caso contrário, os dados serão arremessados novamente.

```
20 LET A=1
30 REM..PRIMEIRO  LANCAMENTO..
40 GOSUB 150
50 LET T1=T
60 REM..SEGUNDO LANCAMENTO..
70 GOSUB 150
80 LET T2=T
90 IF T1=T2 THEN GOTO 120
100 LET A=A+1
110 GOTO 40
120 PRINT"CONTAGENS IGUAIS A ";
T1;" APOS ";A;" LANCAMENTOS"
130 END
140 REM..SUBROTINA
150 LET D1=RND(6)
160 LET D2=RND(6)
170 LET T=D1+D2
180 RETURN
```

Digite o programa abaixo usando somente maiúsculas, se o seu computador for compatível com o ZX-81.

```
20 LET a=1
```

```
 30 REM primeira jogada
 40 GOSUB 150
 50 LET t1=T
 60 REM segunda jogada
 70 GOSUB 150
 80 LET t2=T
 90 IF t1=t2 THEN  GOTO 120
100 LET a=a+1
110 GOTO 40
120 PRINT "Os dados deram ";t1
;" na jogada";a
130 GOTO 200
140 REM sobroutine
150 LET d1=INT (RND*6)+1
160 LET d2=INT (RND*6)+1
170 LET T=d1+d2
180 RETURN
```

```
10 CLS: LET D1=RND(-TIME)
20 LET A=1
40 GOSUB 150
50 LET T1=T
70 GOSUB 150
80 LET T2=T
90 IF T1=T2  THEN GOTO 120
100 LET A=A+1
110 GOTO 40
120 PRINT "CONTAGENS IGUAIS A "
T1;" APOS ";A;" LANCAMENTOS"
130 END
140 REM ... SUBROTINA
150 LET D1=INT(RND(1)*6)+1
160 LET D2=INT(RND(1)*6)+1
170 LET T=D1+D2
175 PRINT D1;D2;T
180 RETURN
```

O arremesso do dado deve ser realizado duas vezes; por isso, é entregue a uma sub-rotina, que vai das linhas 150 a 180. O **GOSUB** nas linhas 40 e 70 envia o computador ao ponto de início da sub-rotina (linha 150). O **RETURN** na linha 180 manda o micro voltar à linha 50, caso o chamado à sub-rotina tenha partido do **GOSUB** da linha 40, ou à linha 80, se tiver partido da linha 70. A linha 140 é usada para identificar o início da sub-rotina, usando a declaração **REM**, que significa apenas um comentário feito pelo programador.

Note a declaração **END** na linha 130 de todos os programas (exceto os do Sinclair, que usam um **GOTO**). Se ela não estivesse aí, o computador passaria a executar a sub-rotina diretamente e, ao atingir a declaração **RETURN**, assinalaria um erro, pois não haveria indicação prévia sobre para que linha retornar.

NÚMEROS DE LINHA INEXISTENTES

Como os micros da linha Sinclair não têm declaração **END**, utilizamos um **GOTO** seguido de um número de linha que não existe no programa, ou seja, 200.

```
130 GOTO 200
```

Em outros micros, isto daria uma mensagem de erro. O mesmo, entretanto, não acontece com a linha Sinclair: ao perceber que a linha 200 não existe, o computador interromperá providencialmente o programa, pronto para começar de novo.

Para tais micros, é melhor usar esses expedientes do que colocar

```
130 STOP
```

que é aceito pelo computador, mas que gera uma mensagem de advertência na tela. Isso poderia levar o usuário a acreditar que houve um erro de execução do programa, especialmente porque a linha 130 não é a última *linha física* do programa, pois está no meio dele, antes da sub-rotina.

Mas seja o mais cuidadoso possível quando utilizar, nesses computadores, o comando **GOTO** seguido por um número de linha inexistente.

Após **GOTO 200**, por exemplo, os micros da linha Sinclair pesquisarão todas as linhas seguintes. E se encontrarem uma instrução em alguma delas, eles a executarão (ou tentarão executá-la), independentemente do conteúdo dessa instrução. Portanto, para ficar mais seguro, o melhor é enviar o computador para a última linha física possível, que pode ser a de número 9999, nos micros da linha Sinclair.

E, para que não haja nenhuma confusão quanto ao que você está fazendo, coloque:

```
9999 REM END
```

CHAMADAS MÚLTIPLAS

É possível uma sub-rotina chamar outras, e estas, por sua vez, outras tantas, gerando assim uma estrutura em diversos níveis. Assim como no jogo citado o dado deve ser lançado duas vezes, a sub-rotina de arremesso pode ser constituída como uma sub-rotina dentro de outra, desde que se mude as últimas linhas do programa, da seguinte forma:

```
150 GOSUB 190
155 LET D1=D
160 GOSUB 190
165 LET D2=D
170 LET T=D1+D2
180 RETURN
190 LET D=RND(6)
195 RETURN
```

Digite o programa abaixo usando somente maiúsculas, se o seu computador for compatível com o ZX-81:

```
150 GOSUB 190
155 LET d1=d
160 GOSUB 190
165 LET d2=d
170 LET t=d1+d2
180 RETURN
190 LET d=INT (RND*6)+1
195 RETURN
```

```
150 GOSUB 190
155 LET D1=D
160 GOSUB 190
165 LET D2=D
170 LET T=D1+D2
180 RETURN
190 LET D=INT(RND(1)*6)+1
195 RETURN
```

Aqui, as linhas 150 e 160 enviam o computador para a sub-rotina na linha 190, que faz o dado rolar (isto é, sorteia um número aleatório inteiro, entre 1 e 6), enquanto as linhas 155 e 165 armazenam os resultados de cada lançamento separado.

Por uma questão de concisão, o computador realiza apenas o lançamento de um dado, e não de dois, separadamente. Veja a seguir:

```
150 LET T=RND(6)+RND(6)
```

```
150 LET T=INT(6*RND+1) +
    INT(6*RND+1)
```

```
150 LET T=INT(6*RND(1)+1)+
    INT(6*RND(1)+1
```

UTILIZE O COMANDO ON...GOSUB

Assim como **GOTO**, a declaração **GOSUB** pode ser utilizada para chamadas complexas a um determinado número de sub-rotinas, começando em linhas diferentes. Nos microcomputadores da linha Sinclair (compatíveis com Spectrum e ZX-81), o **GOSUB** pode ser seguido de uma expressão numérica, ou nome de variável, exatamente como vimos anteriormente para o **GOTO**.

No caso dos computadores de outras linhas, como TRS-80, Apple e MSX, existe a declaração **ON...GOSUB**, que pode ser utilizada para o mesmo efeito.

ORGANIZE AS SUAS COLEÇÕES (2)

- COMO PESQUISAR E MODIFICAR UM REGISTRO
- IMPRIMA O ARQUIVO
- UTILIZE O PROGRAMA PARA SEUS ARQUIVOS

Encontrar o nome e o endereço de uma pessoa da qual se sabe apenas o sobrenome e a cidade em que reside é tarefa simples para o computador. Mas, para isso, é preciso programá-lo com as informações adequadas.

Uma das grandes vantagens do arquivamento no computador, em comparação com métodos mais tradicionais (como um fichário, por exemplo), é a facilidade e rapidez com que se pode encontrar as informações arquivadas.

O programa da lição anterior da série *Aplicações* fornecia os meios para se implementar um sistema altamente flexível de arquivamento no computador. Neste artigo mostramos como se pode pesquisar arquivos, tendo em vista objetivos como a correção de registros desatualizados ou incorretos, ou a anulação de registros desnecessários.

COMO PESQUISAR UM ARQUIVO

A opção de pesquisa é a de número 4 no menu principal. Ela permite que você pesquise os registros a partir da informação contida em um campo.

Se você pressionar a tecla 4, o computador mostrará, no alto da tela, os nomes dos campos que você definiu anteriormente para o arquivo; em seguida, ele perguntará qual campo você quer pesquisar — 1 para o primeiro, 2 para o segundo, 3 para o terceiro e assim por diante, contando a partir do alto da tela. A pergunta seguinte será sobre o que você quer pesquisar nesse campo. Em resposta, você deve digitar a palavra ou o número que quer procurar: JOÃO ou CAÇAROLA, por exemplo, ou BÍBLIA, ou seja lá o que for. Em seguida pressione <**RETURN**> ou <**ENTER**>.

A palavra ou o número que você digitar deve ser exatamente igual ao que está escrito no campo. Se uma palavra foi armazenada com letras maiúsculas nos registros e você estiver procurando-a com letras minúsculas, o computador não irá encontrá-la. Se você tiver deixa-

do, por engano, um espaço antes da entrada em um registro, ou apertar o espacejador após a entrada, provavelmente o computador não irá encontrá-la. Isso ocorre mesmo que a palavra do registro e a que você está procurando sejam iguais.

Assim, evite espaços desnecessários e procure entrar informações em letras maiúsculas. A pesquisa se tornará mais fácil e menos sujeita a erros.

Caso o computador não consiga achar nenhum registro com a palavra que você pediu, no campo pesquisado, ele lhe dirá isso e trará de volta o menu principal. Se, ao contrário, forem encontradas diversas respostas, ele colocará na tela o primeiro registro que tiver a informação pedida e, a partir daí, listará em ordem alfabética todos aqueles com informação igual.

Você deve então selecionar uma das opções colocadas nas duas linhas abaixo da tela. A primeira é assim:

```
PROSSEGUE    RETORNA    MENU
```

Isso funciona mais ou menos da mesma maneira que no programa do artigo anterior. A tecla P serve para pesquisar os registros selecionados em ordem crescente, ao passo que a tecla M o leva diretamente de volta ao menu principal. Mas não acontecerá nada se você pressionar a tecla P quando estiver no último registro selecionado. A tecla R, na primeira modalidade de pesquisa, fornece uma mensagem: "REGISTRO NÃO ENCONTRADO".

Talvez a aplicação mais útil da opção de pesquisa seja descobrir um determinado registro. Para localizá-lo, basta lembrar-se de qualquer informação sobre ele. Por exemplo, suponhamos que você esteja utilizando esse arquivo para armazenar nomes, endereços e números de telefones dos membros de seu clube e já possua os seguintes campos: nome, sobrenome, rua, cidade, Estado,código postal, número do telefone. Neste caso, você poderá pesquisar um determinado registro até mesmo se lembrar apenas do sobrenome da pessoa.

E se uma parte da informação de que você se lembrou não for exclusiva desse registro? A solução é simples: por exemplo, se você procura o registro de uma pessoa, sabendo apenas que ela mora na cidade de São Paulo, ou que o seu primeiro nome é João, poderá evocar todos os paulistanos ou todos os indivíduos de nome João e saltá-los até encontrar o nome correto. Mesmo que o número de nomes fosse muito grande, essa procura seria mais fácil do que se tivesse que percorrer todo o arquivo, revisando um registro de cada vez.

COMO MODIFICAR UM REGISTRO

A segunda linha de opções da parte inferior da tela permite outras opções de trabalho com o registro exibido:

```
CORRIGE    APAGA    IMPRIME
```

Se você quiser modificar um registro mostrado na tela, deve pressionar a tecla C. O computador perguntará "QUAL O NÚMERO DO CAMPO" que você deseja modificar. Digite o número, contando a partir do alto.

Em seguida, o computador pedirá para "ENTRAR A MODIFICAÇÃO". Você deve então digitar o novo campo que quer entrar, até mesmo se quiser modificar apenas uma letra. Quando as teclas <**ENTER**> ou <**RETURN**> forem pressionadas, o computador efetuará a modificação no lugar correto do registro. Se você quiser, poderá pressionar mais uma vez a tecla C e modificar ou-

tro campo no mesmo registro. Se você quiser modificar um registro que não está na tela, deve pressionar C para retornar ao menu principal e escolher a opção de pesquisa 3, para localizar o registro que deseja alterar. Outra saída seria escolher a opção 2, "VER OS REGISTROS", achando-o e modificando-o.

APAGUE OS REGISTROS INCORRETOS

Você deve primeiro localizar o registro que deseja apagar, estudando as alternativas. Normalmente, são apresentadas, no lado inferior da tela, seis opções padrão. Para apagar um registro é necessário pressionar a tecla A. Então o computador perguntará: "VOCÊ TEM CERTEZA?". Isso evita que você apague, por engano, um registro correto.

Se você tem certeza de que quer apagá-lo, pressione a tecla S. O computador apagará então esse registro e exibirá o próximo — tanto o registro anotado segundo uma ordem alfabética, se você estiver na modalidade de exibição, quanto o que possui o mesmo campo que você está pesquisando, se estiver na modalidade de pesquisa.

Se você apagou o último registro encontrado na pesquisa, o computador informará que não existem mais registros com tal item no campo que o interessa e trará o menu principal de volta.

IMPRIMA OS REGISTROS

A última opção na parte inferior da tela permite mandar para a impressora (caso haja uma) os registros do arquivo. Se você pressionar a tecla I na versão do programa para o TRS-Color, o computador lhe pedirá para "CHECAR A IMPRESSORA". Nesse ponto, é necessário averiguar se a impressora está conectada e ligada; se essa condição for satisfeita e mesmo assim a impressora não funcionar, você deverá verificar o programa.

No Spectrum essa linha de mensagem não é necessária, pois ele não reagirá à tecla I se não houver nenhuma impressora conectada.

Uma vez que você já checou se a impressora está conectada e ligada, acione a tecla C para continuar. Pressionando algumas outras letras, a linha "CHEQUE A IMPRESSORA" desaparecerá e você poderá novamente utilizar uma das seis opções padrão, localizadas no lado inferior da tela. Caso C seja pressionada sem que haja uma impressora conectada, ou esta não funcione por qualquer razão, você deve pressionar as teclas **<RESET>** (no TRS-Color e Apple), ou **<CTRL>** **<STOP>** (no MSX), e verificar o programa. Para continuar a utilizar o programa sem perder os dados armazenados, digite de modo direto:

```
GOTO 30
```

Retornando ao menu principal (uma observação importantíssima: não tente modificar qualquer linha do programa antes de dar o comando acima, caso contrário perderá todos os dados já armazenados na memória RAM da máquina).

SAIBA UTILIZAR SEUS ARQUIVOS

Agora você possui um sistema de arquivamento altamente flexível e completo, que pode ser utilizado para armazenar detalhadamente qualquer tipo de informação. Evidentemente, você pode utilizar o programa para armazenar, em uma única fita cassete, um fichário enorme ou vários fichários.

Fitas muito longas, contudo, apresentam inconvenientes. Uma solução mais prática consiste em armazenar diferentes arquivos em várias fitas, de modo a torná-los mais facilmente localizáveis. Neste caso, é necessário guardar o próprio programa em uma fita separada (somente os micros da linha Spectrum estão dispensados dessa precaução). Você poderá, assim, carregar os dados antes de selecionar a opção 6.

À primeira vista, isso pode parecer um grande desperdício de número de acessos. Essa impressão, contudo, deixa de ter sentido quando pensamos no espaço ocupado pelos arquivos dos antigos cartões de indexação.

COMO ENTRAR O PROGRAMA

A seguir, listamos a segunda seção do programa de arquivamento de dados que, para sua felicidade, é muito mais curta do que a primeira.

Quando acrescentá-la ao programa existente, você notará que poucos números de linhas estão duplicados. Não esquente a cabeça com isso. Os números da primeira seção estavam ali simplesmente para manter o programa intacto e permitir o seu teste. Os novos números substituirão os antigos assim que você digitá-los.

T

Atenção se você for utilizar o programa abaixo em micros com disquete. Mude o número 11000 do comando **CLEAR**, no começo do programa, para 8000, sob pena de não haver espaço suficiente na memória.

```
3000 GOSUB 8500
3010 PRINT @417,"MODIFICAR QUAL
 CAMPO?(1 A";A;") ?";
3020 IN$=INKEY$:IF IN$="" THEN
3020
3030 J=VAL(IN$)
3040 IF J<1 OR J>A THEN 3020
3050 PRINT @448,""
3060 PRINT @45+32*J,"":PRINT @4
17,"DIGITE O CAMPO MODIFICADO "
:LINE INPUT A$(D,J)
3070 A$(D,J)=LEFT$(A$(D,J),A(J)
)
3080 IF D=R THEN J=-1:GOTO 3130
3090 IF D=1 THEN J=1:GOTO 3120
3100 IF A$(D,1)>A$(D+1,1) THEN
J=1
3110 IF A$(D,1)<A$(D-1,1)THEN J
=-1
3120 IF A$(D+1,1)="" AND J=1 TH
EN 3180
3130 IF J=1 THEN 3160
3140 IF A$(D,1)>=A$(D-1,1) THEN
 3180
3150 FOR N=1 TO A:X$=A$(D,N):A$
(D,N)=A$(D-1,N):A$(D-1,N)=X$:NE
XT:D=D-1:GOTO 3080
3160 IF A$(D,1)<=A$(D+1,1) THEN
 3180
3170 FOR N=1 TO A:X$=A$(D,N):A$
(D,N)=A$(D+1,N):A$(D+1,N)=X$:NE
XT:D=D+1:GOTO 3080
3180 FOR F=1 TO 300:NEXT:RETURN
4000 G=D
4010 GOSUB 8500
4020 PRINT @449,"VOCE TEM CERTE
ZA QUE QUER APAGAR?"
4030 IN$=INKEY$:IF IN$="" THEN
4030
4040 IF IN$<>"S" THEN RETURN
4050 IF G=NR THEN G=G-1
4060 CLS 7:PRINT @236,"APAGANDO
";:SOUND 30,1
4070 IF D=NR THEN 4090
4080 FOR N=1 TO A:A$(D,N)=A$(D+
1,N):NEXT:D=D+1:GOTO 4070
4090 FOR N=1 TO A:A$(D,N)="":NE
XT:NR=NR-1:D=G
4100 FOR F=1 TO 400:NEXT:RETURN
5000 FOR N=1 TO A:PRINT @33+32*
N,N,N$(N):NEXT
5010 PRINT @417,"QUE CAMPO PROC
URAR (1 A";A;") ?";
5020 IN$=INKEY$:IF IN$="" THEN
5020
5030 IF VAL(IN$)<1 OR VAL(IN$)>
A THEN 5020
5040 SOUND 30,1:Z=VAL(IN$):PRIN
T @417,"PROCURAR PELO QUE (NO C
AMPO";Z;")?"
5050 LINE INPUT Z$
5060 D=1:G=0:CH=1
5070 IF D>NR AND G=1 THEN G=0:C
H=-1 ELSE IF D>NR THEN 5230
5080 IF D<1 AND G=1 THEN G=0:CH
=1 ELSE IF D<1 THEN 5230
5090 IF A$(D,Z)<>Z$ THEN D=D+CH
:GOTO 5070
5100 LD=D:G=1:GOSUB 8500
5110 PRINT @451,"pROSSEGUE
```

```
ETORNA    mENU    cORRIGE    aPA
GA     iMPRIME";
5120 IN$=INKEY$:IF IN$="" THEN
5120
5130 IN=INSTR(1,RS$,IN$)
5140 ON IN GOTO 5160,5160,5170,
5180,5190,5200,5210
5150 GOTO 5120
5160 CH=1:D=D+1:GOTO 5070
5170 CH=-1:D=D-1:GOTO 5070
5180 RETURN
5190 GOSUB 3000:GOTO 5090
5200 GOSUB 4000:GOTO 5090
5210 GOSUB 10000:GOTO 5070
5220 GOTO 5110
5230 CLS 2:PRINT @200,"NENHUM R
EGISTRO COM ";:PRINT @271-LEN(Z
$)/2," ";Z$;" ";
5240 PRINT @330," NO CAMPO";Z;
5250 FOR G=1 TO 5:SCREEN 0,1:FO
R F=1 TO 500:NEXT:SCREEN 0,0:FO
R F=1 TO 500:NEXT F,G:RETURN
```

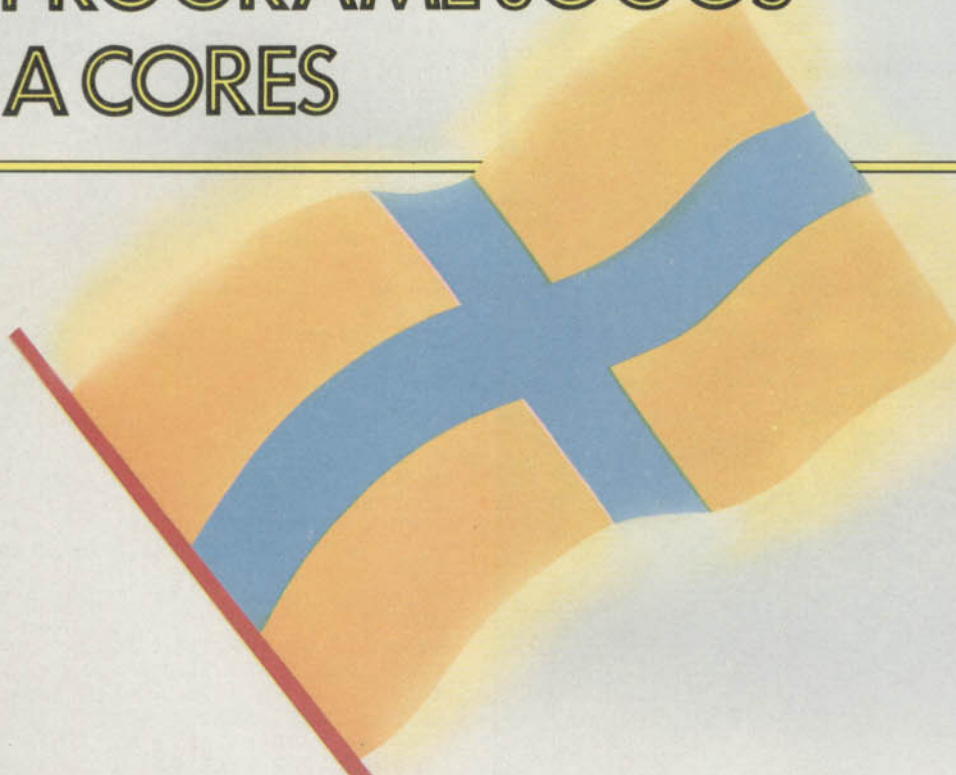
```
4000 FOR N=V TO A: PRINT  INVER
SE V;AT N*2,9;N; INVERSE U;TAB
11;":  ";N$(N): NEXT N
4010 PRINT '''"PROCURAR EM QUE C
AMPO (1 A ";A;")?"
4020 IF INKEY$="" THEN  GOTO 40
20
4030 LET Y$=INKEY$: IF CODE Y$<
49 OR CODE Y$>48+A THEN  GOTO 4
030
4040 SOUND .1,10: LET Z=VAL Y$:
 PRINT '"PROCURAR PELO QUE NO C
AMPO ";Z;"?": DIM Z$(V,A(Z)): I
NPUT  LINE Z$(V)
4044 CLS : LET K=V
4045 IF A$(K,B(Z)+V TO B(Z+V))=
Z$(V) THEN  GOTO 4050
4046 IF K=R OR A$(K,V)=" " THEN
  CLS : PRINT AT 9,3;"NENHUM RE
GISTRO COM ";Z$(V),'TAB 10;"NO
CAMPO ";Z: PAUSE 150: RETURN
4047 LET K=K+V: GOTO 4045
4050 LET D=V: LET PM=V: LET MO=
V
4060 IF D>R THEN  LET D=PM
4070 IF D=U THEN  LET D=PM
4080 IF A$(D,V)=" " THEN  LET D
=PM
4090 IF A$(D,B(Z)+V TO B(Z+V))<
>Z$(V) THEN  LET D=D+MO: GOTO 4
060
4100 GOSUB 9500
4110 LET PM=D
4120 IF OP=V THEN  LET MO=V: LE
T D=D+MO: GOTO 4060
4130 IF OP=2 THEN  LET MO=-V: L
ET D=D+MO: GOTO 4060
4140 IF OP=3 THEN  RETURN
4150 IF OP=4 THEN  GOSUB 8000
4160 IF OP=5 THEN  LET DF=U: LE
T MD=2: GOSUB 9000: IF DF=V OR
A$(V,V)=" " THEN  RETURN
4170 GOTO 4100
8000 INPUT AT U,U;"CAMPO A SER
CORRIGIDO(1 A ";(A);")?";J: IF
J>A THEN  GOTO 8000
8010 PRINT  FLASH V;AT V+2*J,12
;A$(D,B(J)+V TO B(J+V)): INPUT
AT U,U;"Digite o campo modifica
do", LINE A$(D,B(J)+V TO B(J+V)
): PRINT AT V+2*J,12;A$(D,B(J)+
V TO B(J+V))
8020 IF D=R THEN  LET J=-V: GOT
O 8070
8030 IF D=V THEN  LET J=V: GOTO
 8060
8040 IF A$(D)>A$(D+V) THEN  LET
 J=V
8050 IF A$(D)<A$(D-V) THEN  LET
 J=-V
8060 IF A$(D+V,V)=" " AND J=V T
HEN  GOTO 8500
8070 IF J=V THEN  GOTO 8100
8080 IF A$(D)>=A$(D-V) THEN  GO
TO 8500
8090 LET X$=A$(D): LET A$(D)=A$
(D-V): LET A$(D-V)=X$: LET D=D-
V: GOTO 8020
8100 IF A$(D)<=A$(D+V) THEN  GO
TO 8500
8110 LET X$=A$(D): LET A$(D)=A$
(D+V): LET A$(D+V)=X$: LET D=D+
V: GOTO 8020
8500 SOUND .1,10: PRINT AT U,U;
"Registro numero ";D;"   ": RET
URN
9000 PRINT AT 19,U;"TEM CERTEZA
 DE QUE QUER APAGAR?": PAUSE U
9010 IF INKEY$="" THEN  GOTO 90
10
9020 IF INKEY$<>"S" THEN  PRINT
 AT 19,U;"
          ": SOUND .1,10: RETU
RN
9030 PRINT AT 19,U;" APAGANDO
                          "
9035 LET DD=D
9040 IF D=R THEN  LET DD=DD-V:
GOTO 9060
9050 IF A$(D+V,V)<>" " THEN  LE
T A$(D)=A$(D+V): LET D=D+V: GOT
O 9040
9060 LET A$(D)="": FOR F=V TO 1
00: NEXT F: PRINT AT 19,U;"
     "
9070 LET D=DD: IF A$(V,V)=" " T
HEN  LET D=U: RETURN
9072 IF D=U THEN  LET D=V
9075 IF A$(D,V)=" " THEN  LET D
=D-V
9080 IF MD=V THEN  RETURN
9090 LET K=V
9100 IF A$(K,B(Z)+V TO B(Z+V))=
Z$(V) THEN  GOTO 9130
9110 IF K=R OR A$(K,V)=" " THEN
  LET DF=V: GOTO 4046
9120 LET K=K+V: GOTO 9100
9130 LET DD=D: LET PA=V
9140 IF A$(DD,V)=" " OR DD=U TH
EN  LET PA=2: LET DD=D: LET MO=
MO-V
9150 IF A$(DD,B(Z)+V TO B(Z+V))
=Z$(V) THEN  LET D=DD: RETURN
9160 LET DD=DD+MO: GOTO 9140
```

```
3000 GOSUB8500
3010 LOCATE0,22:PRINT"Modificar
 qual campo? (1 até ";A;")";:
3020 IN$=INKEY$:IFIN$=""THEN302
```

```
0
3030 J=VAL(IN$)
3040 IFJ<1ORJ>ATHEN3020
3060 LOCATE0,22:PRINTSPACE$(39)
;:LOCATE0,22:PRINT"Entre com o
campo modificado: ";:LINEINPUTA
$(D,J)
3070 A$(D,J)=LEFT$(A$(D,J),A(J)
)
3080 IFD=RTHENJ=-1:GOTO3130
3090 IFD=1THENJ=1:GOTO3120
3100 IFA$(D,1)>A$(D+1,1)THENJ=1
3110 IFA$(D,1)<A$(D-1,1)THENJ=-
1
3120 IFA$(D+1,1)=""ANDJ=1THEN31
80
3130 IFJ=1THEN3160
3140 IFA$(D,1)>=A$(D-1,1)THEN31
80
3150 FORN=1TOA:SWAPA$(D,N),A$(D
-1,N):NEXT:D=D-1:GOTO3080
3160 IFA$(D,1)<=A$(D+1,1)THEN31
80
3170 FORN=1TOA:SWAPA$(D,N),A$(D
+1,N):NEXT:D=D+1:GOTO3080
3180 FORF=1TO300:NEXT:RETURN
4000 G=D
4010 GOSUB8500
4020 LOCATE0,22:PRINT"Você conf
```

```
irma a deleção? (S/N)";
4030 IN$=INKEY$:IFIN$=""THEN403
0
4040 IFIN$<>"S"THENRETURN
4050 IFG=NRTHENG=G-1
4060 CLS:LOCATE12,15:PRINTCHR$(
7);"Apagando";CHR$(7);
4070 IFD=NRTHEN4090
4080 FORN=1TOA:A$(D,N)=A$(D+1,N
):NEXT:D=D+1:GOTO4070
4090 FORN=1TOA:A$(D,N)="":NEXT:
NR=NR-1:D=G
4100 FORF=1TO400:NEXT:RETURN
5000 FORN=1TOA:LOCATE5,2*N+2:PR
INTN,N$(N):NEXT
5010 LOCATE0,22:PRINT"Procurar
por qual campo? (1 até ";A;")";
5020 IN$=INKEY$:IFIN$=""THEN502
0
5030 IFVAL(IN$)<1ORVAL(IN$)>ATH
EN5020
5040 PRINTCHR$(7);:Z=VAL(IN$):L
OCATE0,22:PRINT"Procurar o que
no campo ";Z;"?";SPACE$(8)
5050 LINEINPUTZ$
5060 D=1:G=0:CH=1
5070 IFD>NRANDG=1THENG=0:CH=-1E
LSEIFD>NR1ЧEN5230
5080 IFD<1ANDG=1THENG=0:CH=1ELS
EIFD<1THEN5230
5090 IFA$(D,Z)<>Z$THEND=D+CH:GO
TO5070
5100 LD=D:G=1:GOSUB8500
5110 LOCATE3,22:PRINT"[P]rosseg
ue    [R]etorna    [M]enu    [C]o
rrige     [A]paga       [I]mprime"
;
5120 IN$=INKEY$:IFIN$=""THEN512
0
5130 IN=INSTR(1,R$,IN$)
5140 ONINGOTO5160,5160,5170,518
0,5190,5200,5210
5150 GOTO5120
5160 CH=1:D=D+1:GOTO5070
5170 CH=-1:D=D-1:GOTO5070
5180 RETURN
5190 GOSUB3000:GOTO5090
5200 GOSUB4000:GOTO5090
5210 GOSUB10000:GOTO5070
5220 GOTO5110
5230 CLS:LOCATE0,15:PRINT"Não e
ncontrei ";Z$;" em ";N$(Z)
5250 FORF=1TO500:NEXT:RETURN
```

```
3000  POKE 34,22: GOSUB 6500
3010  VTAB 23: CALL  - 958: PRI
NT "NUMERO DO CAMPO A SER MODIF
ICADO ->";: GET CP$
3020  IF CP$ < "0" OR CP$ > "8"
 THEN 3010
3030 CP =  VAL (CP$): IF CP = 0
 THEN  POKE 34,0: RETURN
3040  VTAB 23: HTAB 1: CALL  -
958
3050  PRINT "CAMPO CORRIGIDO ->
";: INPUT A$(D,CP)
3060 A$(D,CP) =  LEFT$ (A$(D,CP
),A(CP))
3070  IF CP <  > 1 THEN 3000
3080  IF A$(D,1) > A$(D + 1,1)
AND D <  > NR THEN V = 1: GOTO
3200
3090  IF A$(D,1) < A$(D - 1,1)
AND D > 0 THEN V =  - 1: GOTO 3
200
3100  GOTO 3000
3200  FOR N = 1 TO A:X$ = A$(D,
N):A$(D,N) = A$(D + V,N):A$(D +
 V,N) = X$: NEXT :D = D + V: GO
TO 3080
4000  GOSUB 6500: VTAB 23
4010  PRINT "CONFIRMA A DELECAO
? ";: GET IN$
4020  IF IN$ <  > "S" THEN  RET
URN
4030  IF D = NR THEN 4060
4040  FOR X = D TO NR
4050  FOR Y = 1 TO A:A$(X,Y) =
A$(X + 1,Y): NEXT : NEXT
4060 NR = NR - 1:D = NR: IF D =
 0 THEN  POP : RETURN
4070  RETURN
5000 DD = 1: PRINT : FOR X = 1
TO A: PRINT : HTAB 10: PRINT X;
":-";N$(X): NEXT
5010  VTAB 21: PRINT "PROCURAR
POR QUAL CAMPO? ";: GET CP$
5020 CO =  VAL (CP$): IF CO > A
 OR CO < 1 THEN  RETURN
5025  PRINT CO
5030  VTAB 22: HTAB 1: PRINT "P
ROCURAR O QUE EM ";N$(CO);: INP
UT PR$
5035  IF PR$ = "" THEN  RETURN
5040  HTAB 13: PRINT "PROCURAND
O...";
5050  FOR XX = DD TO NR: IF  LE
FT$ (A$(XX,CO), LEN (PR$)) = PR
$ THEN FL = 1:D = XX: GOSUB 602
0: GOTO 5200
5060  NEXT
5070  IF FL = 0 THEN  VTAB 21:
HTAB 1: CALL  - 958: PRINT "NAO
 ENCONTREI ";PR$;" EM ";N$(CO):
 FOR X = 1 TO 5000: NEXT : GOTO
 5090
5080  VTAB 21: HTAB 1: CALL  -
958: HTAB 8: PRINT "FIM DE ARQU
IVO ENCONTRADO": FOR X = 1 TO 5
000: NEXT
5090 FL = 0:PR$ = "": RETURN
5200  IF XX = NR THEN 5090
5210  VTAB 15: PRINT "CONTINUO
PROCURANDO ";PR$: PRINT " EM ";
N$(CO);"? (S/N) ";: GET IN$
5220  IF IN$ = "N" THEN 5090
5230  IF IN$ = "S" THEN DD = XX
 + 1: GOTO 5050
5240  GOTO 5210
```

PROGRAME JOGOS A CORES

O TRS-Color (e seus compatíveis nacionais, como o CP-400) possui cinco modalidades para elaboração de gráficos de alta resolução (**PMODE**), numerados de 0 a 4. Eles diferem na resolução, no grau de detalhe e nas cores dos desenho exibidos. **PMODE** é uma instrução BASIC, específica para esses micros, e significa *picture modes* (modalidades gráficas). Ela estabelece o conjunto de cores a ser empregado. Na modalidade de quatro cores — **PMODE** 1 ou 3 —, o conjunto 0 consiste de verde, amarelo, azul e vermelho, enquanto o conjunto 1 de cores é formado por branco, azul-piscina (ciano), magenta e laranja. Para o conjunto 0 de cores, digite o comando **SCREEN 1,0**; para o conjunto 1, recorra a **SCREEN 1,1.**

A diferença entre **PMODE 1** e **PMODE 3** se encontra na resolução da tela gráfica que eles utilizam. Resolução gráfica é o número de *pixels* (*picture element* ou elemento de figura) que podem ser traçados na tela e, como o nome diz, é a unidade mínima de construção de figuras. Ele corresponde a um bloco retangular "aceso" na tela.

O programa abaixo mostra a capacidade de trabalho com gráficos de resolução média do TRS-Color. Do jeito que está, ele roda apenas em máquinas *com gravador cassete*. Para executá-lo em uma máquina com Disk BASIC, altere os endereços nos comandos **POKE** nas linhas 30 e 50, mudando-os de 1800 e 1801 para 3584 e 3585, respectivamente.

```
10 PMODE3,1:PCLS:SCREEN1,0
20 FOR I=1 TO 9
30 READ A,B:POKE 1800+I*32,A:PO
KE 1801+I*32,B
40 NEXT
```

Você já sabe como construir blocos gráficos em preto e branco. Aprenda agora a colorir seus desenhos em micros da linha TRS-Color, empregando técnicas mais sofisticadas.

- COMO UTILIZAR CONJUNTOS DE CORES DIVERSAS
- COMO DEFINIR CORES EM BLOCOS GRÁFICOS
- EXPLORE OS **PMODE**

```
50 FOR K=10 TO 22:POKE 1800+K*3
2,192:POKE 1801+K*32,0:NEXT
60 GOTO 60
70 DATA 192,0,213,149,213,149,2
13,149,234,170,234,170,213,149,
213,149,213,149
```

O programa desenha uma bandeira e um mastro na tela. Os códigos dos blocos gráficos que constituem o desenho são especificados em uma linha **DATA**; o programa as lê e coloca os caracteres gráficos correspondentes na memória de vídeo, através do comando **POKE** ou **PRINT@**. Desta vez, para que você possa ver os blocos gráficos mais claramente, os códigos são colocados na página de memória de vídeo, a partir do endereço 1800, o que os situa no meio da tela. Isto ocorre porque a página gráfica usada (**PMODE 3**) tem seu endereço absoluto de início na locação 1536. O programa não funcionará em máquinas com BASIC de disco.

A linha 10 seleciona o **PMODE 3** (conjunto de cores 0) e limpa a tela. As linhas 20 a 40 lêem nove pares de dados que definem o desenho da bandeira, e os colocam diretamente na página de vídeo correspondente, através do comando **POKE**. O restante da bandeira é desenhado pela linha 50, que traça o mastro através de dois **POKE** (códigos 192 e 0), repetidos 22 vezes. A linha 60 forma um laço sem fim, cuja única função é "congelar" a tela gráfica. Pressionando **<BREAK>**, você interrompe o programa.

A principal diferença entre a técnica que tínhamos aprendido antes e a utilizada neste programa está nos dados de definição da figura. Os códigos estão em decimal e foram convertidos a partir de um número binário de oito bits. Mas, ao invés de bits individuais, que dizem ao computador para ligar ou desligar pixels individuais, os pares de bits estabelecem agora que cor será atribuída aos pares de pixels. No **PMODE 1**, um par de bits diz ao computador para situar dois pares de pixels, um abaixo do outro na tela.

A figura 1 mostra como a bandeira é feita de pares de pixels: ela é definida através de um conjunto de números binários de dois bits, correspondentes a cada um dos dois blocos de pixels. No conjunto de cores 0 você deve escrever 00 para um bloco de cor verde, 01 para um bloco amarelo, 10 para um azul e 11 para um vermelho. Quatro desses números de dois bits compõem cada parte dos dados de oito bits.

Caso tenha escolhido o conjunto de cores 1, digitando **SCREEN 1,1** na linha 10, você terá uma bandeira desenhada em branco, ciano, magenta e laranja. Nesse conjunto de cores você escreve 00 quando quer um bloco branco opaco, 01 para um azul ciano, 10 para um bloco magenta e 11 para um laranja.

O programa abaixo permite ver o efeito que os códigos gráficos em **DATA** têm sobre os pixels e as suas cores em cada um dos **PMODE** (rodar apenas nos micros com BASIC para cassete).

```
10 CLS
20 INPUT"MODO GRAFICO(0-4) ";MO
30 MO=INT(MO):IF MO<0 OR MO>4 T
HEN 20
40 INPUT"NUMERO DA TELA(0-1) ";
ST
50 ST=INT(ST):IF ST<0 OR ST>1 T
HEN 40
60 INPUT "NUMERO(0-255) ";NU
70 NU=INT(NU):IF NU<0 OR NU>255
 THEN 60
80 IF MO<4 THEN LE=2 ELSE LE=1
90 IF MO<2 THEN DE=2 ELSE DE=1
100 IF(MO AND 1)=1 THEN SP=-2 E
LSE SP=-1
110 FOR K=7 TO 0 STEP SP
120 FOR L=1 TO LE
130 FOR J=1 TO DE
140 IF SP=-1 THEN PP=1358+32*J+
(3-K)*LE+L:CO=INT(.5+(NU AND 2^
K)/2^K):GOTO 160
150 PP=1393+32*J+L-K:CO=INT(.5+
(NU AND 3*2^(K-1))/2^(K-1))
160 POKE PP,113-SP*15+CO*(14-SP
)+ST*64
170 NEXT J,L
180 POKE PP-32*DE,47+K
190 POKE PP-32*DE+1+SP,48+K
200 IF SP=-2 THEN POKE PP+31,11
2-(CO>1)
210 POKE PP+21,112+(CO AND 1)
220 NEXT K
230 PRINT @482,"PRESSIONE QUALQ
UER TECLA PARA CONTINUAR";
240 SCREEN 0,1-ST
250 A$=INKEY$:IF A$="" THEN 250
260 GOTO 10
```

O programa lhe pedirá para selecionar uma modalidade de gráfico (**PMODE**) e o número de tela (**SCREEN**). A seguir, digite um número entre 0 e 255. Aparecerão na tela as cores e os tamanhos relativos dos pixels, com os números dos bits em cima e os seus equivalentes binários embaixo.

192	1	1	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
213	1	1	Ø	1	Ø	1	Ø	1	1	Ø	Ø	1	Ø	1	Ø	1	149
213	1	1	Ø	1	Ø	1	Ø	1	1	Ø	Ø	1	Ø	1	Ø	1	149
213	1	1	Ø	1	Ø	1	Ø	1	1	Ø	Ø	1	Ø	1	Ø	1	149
234	1	1	1	Ø	1	Ø	1	Ø	1	Ø	1	Ø	1	Ø	1	Ø	17Ø
234	1	1	1	Ø	1	Ø	1	Ø	1	Ø	1	Ø	1	Ø	1	Ø	17Ø
213	1	1	Ø	1	Ø	1	Ø	1	1	Ø	Ø	1	Ø	1	Ø	1	149
213	1	1	Ø	1	Ø	1	Ø	1	1	Ø	Ø	1	Ø	1	Ø	1	149
213	1	1	Ø	1	Ø	1	Ø	1	1	Ø	Ø	1	Ø	1	Ø	1	149
192	1	1	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
192	1	1	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
	1	1															
	1	1															
	1	1															
	1	1															

Nº DO PMODE	PIXELS	CONJUNTO DE CORES	
		SCREEN 1,0	SCREEN 1,1
0	■■ ■■	preto e verde	preto e branco
1	■■ ■■	verde, amarelo, azul e vermelho	branco, ciano, magenta e laranja
2	■■	preto e verde	preto e branco
3	■■	verde, amarelo, azul e vermelho	branco, ciano, magenta e laranja
4	■	preto e verde	preto e verde

COMO ENTRAR CÓDIGO DE MÁQUINA

- COMO ENCONTRAR UM LUGAR PARA O CÓDIGO DE MÁQUINA
- PROGRAMA MONITOR
- SAIBA USAR O MONITOR
- RODE UM PROGRAMA

O micro responde automaticamente a comandos em BASIC. Nesta lição você aprenderá a trabalhar com código de máquina, usando um programa em BASIC e a instrução POKE.

Embora o código de máquina seja a linguagem de microprocessador, não existem meios de colocá-lo diretamente na memória de um computador pessoal. É preciso entrá-lo através de um outro programa — o que na maioria dos micros atuais significa utilizar a linguagem BASIC para esse programa, pois ela é a única disponível na memória ROM da máquina. Além disso, não podemos mandar o computador executar um programa em código de máquina utilizando um comando nesse código: é preciso utilizar também uma instrução em BASIC.

Existem programas especiais para quem deseja programar diretamente em linguagem de máquina (ou seja, utilizando códigos em hexadecimal). Esses programas são chamados de *monitores* e permitem, entre outras coisas, entrar um programa em hexadecimal na memória da máquina, executá-lo, listá-lo na tela ou na impressora, corrigi-lo, etc.

Os micros da linha TRS-80, compatíveis com as máquinas de Radio Shack americanas, modelos III e IV, têm um monitor disponível na memória. Esse monitor permite o trabalho direto com códigos hexadecimais e é bastante completo. Já os micros compatíveis com o Modelo I do TRS-80 não dispõem desse recurso, necessitando de um programa elaborado em BASIC, especial ou disponível comercialmente.

Os micros das linhas Apple II+, IIe e IIc, bem como o Microdigital TK-2000, em todas as suas versões, têm um programa monitor embutido, para trabalho direto com hexadecimais. O modelo TK-2000 conta ainda com uma vantagem adicional: um mini-Assembler disponível na ROM do equipamento, que permite a programação com códigos mnemônicos e que pode ser invocado diretamente do BASIC, pelo comando **MA**.

Essas máquinas não oferecem monitores ou montadores embutidos na ROM. Precisam, portanto, de programas externos, que devem ser carregados previamente na memória RAM, para o trabalho direto com a linguagem de máquina (códigos hexadecimais).

O instrumento básico para entrar uma rotina de máquina é o comando **POKE**. Ele serve para colocar os códigos de máquina, byte por byte, na memória do computador, usando *endereços absolutos* (isto é, o número real da locação de memória, em vez de um nome simbólico de variável, como o Assembler e o BASIC fazem).

Nos micros compatíveis com Sinclair Spectrum, ZX-81, e TRS-80 Modelo I (BASIC Nível II, da versão cassete), você não pode usar um hexadecimal com o comando **POKE**, apenas um número decimal. Assim, os hexadecimais de sua rotina de código de máquina devem ser convertidos para decimais, antes de serem entrados. Já o TRS-Color, os TRS-80 Mod. III e IV (com o Disk BASIC), o MSX e o Apple II aceitam hexadecimais no **POKE**.

ONDE COLOCAR O PROGRAMA

Antes de entrar um programa em código de máquina, você deve decidir onde irá colocá-lo na memória. Obviamente, não se deve pô-lo em uma área de memória utilizada pelo micro, senão o seu programa será destruído quando algum outro na memória ROM entrar em ação, ou o computador deixará de funcionar corretamente, pois terá algum dado alterado pelo seu programa.

Você também precisa ser cuidadoso quando utilizar as áreas de memória RAM normalmente reservadas pelo interpretador BASIC (essas áreas são descritas no manual de operação ou programação de seu micro). Como a rotina de código de máquina pode ser chamada de um programa em BASIC, se um código de máquina for gravado nessa área enquanto o programa em BASIC estiver sendo rodado, ocorrerão falhas de operação, perdas de programa, ou até o "congelamento" da máquina.

Você pode utilizar áreas de memória intermediária, como o *buffer* da interface para o gravador cassete ou para a impressora, se o programa for curto, e desde que você não esteja utilizando esses periféricos. Se você olhar o manual de programação do seu computador, verá também que alguns modelos têm na memória RAM áreas não usadas que podem ser empregadas para programas curtos em código de máquina.

Nos micros compatíveis com o Sinclair Spectrum, como o Microdigital TK-90X, você pode utilizar a área de gráficos definíveis pelo usuário (UDG) — entre FF58 e FFFF, no modelo de 48 K — desde que ela não esteja sendo utilizada pelo seu programa. Mas, normalmente, programas em código de máquina são colocados entre a área de gráficos UDG e o final da área BASIC, através da modificação do endereço do sistema, chamado RAMTOP (e que identifica o maior endereço disponível na memória RAM do computador), para um valor menor.

Desse modo, a área para o BASIC fica diminuída, mas, como não pode ultrapassar o endereço armazenado na locação RAMTOP, o código de máquina é protegido contra qualquer interferência por parte do BASIC.

A RAMTOP é uma variável de sistema, e sua posição é fornecida pelo apontador nas locações de memória 5CB2 e 5CB3. Para movimentar o RAMTOP para baixo você precisa apenas colocar, com o comando **POKE**, um valor mais baixo nessas duas locações. Mas o Spectrum possui um outro comando BASIC: o **CLEAR**, que faz essencialmente a mesma coisa e deve ser seguido pelo endereço decimal do lugar para o qual você quer que o RAMTOP seja rebaixado.

O comando **CLEAR** também limpa o arquivo de exibição (ou seja, a tela), da mesma forma que um **CLS** (e todas as va-

riáveis). Além disso, ele restaura a posição do comando gráfico **PLOT** para as coordenadas 0,0 — que definem o lado inferior esquerdo da tela —, restabelece o apontador do comando **READ** para o início da lista de declarações **DATA** do programa, e zera todos os **RETURN** ativos. Em um TK-90X com 16 K, por exemplo, o comando mais usual é:

```
CLEAR 31999
```

Isso modifica o RAMTOP para 31 999 ou 7CFF em hexa, deixando 600 bytes acima dele — isto é, protegendo a área que vai desde esse ponto até o final da área de UDG (que está em 32 600) para a colocação de programas em código de máquina. No modelo de 48 K, o comando deve ser, neste exemplo:

```
CLEAR 63999
```

O RAMTOP desce, de tal modo que a área do BASIC passa a terminar em 63 999, ou F999 em hexa: você pode, assim, iniciar o seu programa de código de máquina em 64 000 ou FADO em hexa. Na maioria das aplicações isso deixará espaço suficiente para os seus programas em código de máquina.

Com programas de código de máquina mais longos que o usual, você deve abaixar o RAMTOP ainda mais. Mas isso pode não ser bom na prática: com o RAMTOP tão baixo, não existe espaço nem mesmo para entrar uma linha de BASIC.

No ZX-81, pode-se abaixar o RAMTOP, alterando as locações de memória 16 388 e 16 389 por meio de comandos **POKE**, de modo a criar uma área protegida, onde os programas de código de máquina não sejam modificados. O problema é que não se pode armazenar em fita o programa que estiver nessa área.

Programas curtos em código de máquina podem ser colocados com **POKE** na memória intermediária (*buffer*) para a impressora, que ocupa as locações de memória de 16 444 a 16 476. Existem outras áreas pequenas nas quais você pode colocar seus programas em código de máquina, mas não é possível guardá-los em fita.

Existem três maneiras de se colocar um programa de código de máquina em uma área BASIC, protegendo-o contra outras gravações: pode-se entrá-lo como parte de declarações **REM**; colocá-lo dentro de variáveis alfanuméricas ou cordões de caracteres; ou pode-se armazená-lo em um conjunto numérico.

Para entrar um programa em código de máquina como parte de uma declaração **REM** você deve calcular qual será sua extensão (contar os bytes). Então, na primeira linha do programa em BASIC, digite um **REM** seguido de pontos:

```
10 REM .......................
```

O número de pontos deve ser pelo menos igual ao número de bytes em seu programa de código de máquina.

Em seguida, você deve colocar os códigos do seu programa nas locações de memória ocupadas pela declaração **REM**, um byte de cada vez. Isto é feito por intermédio da declaração **POKE**. Se sua **REM** estiver na primeira linha do programa em BASIC, a área reservada para isso começa na locação de memória 16 514.

Outra maneira de criar um espaço protegido é dimensionar um conjunto. A linha de programa em BASIC:

```
10 DIM A(100)
```

fornecerá 500 locações livres da memória na área de variáveis. Para cada elemento desse conjunto, o ZX-81 deixa cinco locações livres. Se você quiser descobrir onde começar a armazenar o código de máquina neste caso, deverá utilizar a função **PEEK**, para obter o valor armazenado na variável de sistema **VARS**, que está localizada em 16 400 e 16 401:

```
PRINT PEEK 16400+PEEK 16401+6
```

Isso fornece o endereço inicial da área protegida que você criou (as seis locações extras de memória contêm detalhes do conjunto). Você poderá, então, usar a instrução POKE para entrar o programa em código de máquina, a partir do endereço reservado. É possível também criar espaço em um cordão com uma linha BASIC, como:

```
10 LET S$=".................."
```

com tantos pontos quanto bytes no programa de código de máquina. Para achar o endereço inicial, você deve examinar o que tem em **VARS** e acrescentar 6. As seis locações extras de memória, neste caso, contêm o nome do cordão.

O problema com esses dois métodos é que a área das variáveis é apagada quando se aperta a tecla **CLEAR** ou se roda o programa em BASIC de novo.

T

O modo mais fácil de proteger os programas de código de máquina nesse computador consiste em utilizar o comando **CLEAR** do BASIC. O **CLEAR** também zera todas as variáveis numéricas alfanuméricas. Usualmente, ele assume esta forma:

```
CLEAR 200,30000
```

Ele limita o topo da memória RAM (normalmente, em H7FFF ou 32 767 em decimal) em 30 000, deixando 2 767 bytes livres para os programas de código de máquina.

Nesses micros o comando **CLEAR** tem outra função separada, e não relacionada: a atribuição de espaço de memória para as variáveis alfanuméricas. O primeiro número do comando **CLEAR** fornece o espaço de cordões que deve ser reservado para o programa em BASIC. Quando a máquina é ligada, o computador guarda 200 bytes logo abaixo do endereço máximo de memória (RAMTOP) paras as variáveis alfanuméricas. Assim, quando você deixar o RAMTOP mais baixo, deverá reservar, abaixo dele, também uma quantidade similar (200 bytes), para os cordões.

Teoricamente, você pode baixar o RAMTOP do TRS-Color até quase o topo da área reservada para as telas gráficas, embora a máquina vá precisar de um mínimo de espaço para o BASIC.

O **PCLEAR 1** deixa apenas uma das páginas de gráficos; assim, você usa o **CLEAR** para limitar a memória em §H3680 (em computadores sem disco, apenas). O **PCLEAR 8** atribui as oito páginas de gráficos, permitindo que se reserve memória só até §H3680. Se você não utiliza o comando **CLEAR**, o computador atribui quatro páginas de gráficos; assim, você só poderá usar o **CLEAR** até §H1E80.

Esses limites dependem muito do que está dentro da máquina no momento. Se existir algum tipo de programa em BASIC, provavelmente você obterá uma mensagem de erro "OM" (*out of memory*). Assim, para todos os propósitos práticos, trabalhar próximo ao limite não será bom, pois você deverá entrar um programa BASIC em algum lugar para chamar o seu programa de código de máquina. Se você quiser iniciar seu programa de código de máquina com um número redondo como 30 000 então digite:

```
CLEAR 200,29999
```

Para proteger os programas de código de máquina neste micro utilize o comando **CLEAR** do BASIC. Este limita a memória RAM máxima, reservando um espaço acima para seu programa em código de máquina. O **CLEAR** também zera as variáveis numéricas alfanuméricas. Normalmente, ele assume esta forma:

```
CLEAR 100,&HE000
```

Ele limita o topo da memória RAM em 57 344, deixando 8 192 bytes livres para os

2012 85 FD
2010 B1 FB
200E A0 00
200C 85 FE
200A A8 00
RAMTOP
1FF8 55 59
1FF7 2A
1FF6 4F
1FF5 4B
1FF2 2E 4D 4B
1FEF 2C 2F 2C
FE4 4B
FE1 4C 4C
FEØ 4B

programas de código de máquina.

Nestes micros o comando **CLEAR** tem uma outra função separada e não relacionada: a atribuição de espaço de memória para as variáveis alfanuméricas. O primeiro número do comando **CLEAR** fornece o espaço de cordões que deve ser reservado para o programa em BASIC. O computador guarda automaticamente 100 bytes abaixo do endereço máximo de memória (RAMTOP) para as variáveis alfanuméricas quando a máquina é ligada. Se você deixar o RAMTOP artificialmente mais baixo, deverá destinar também uma quantidade similar — 100 bytes — abaixo dele, para os strings.

UM PROGRAMA MONITOR

O programa seguinte é um monitor simples para auxílio à programação em linguagem de máquina. Ele permite que você digite códigos de máquina na memória do computador, guarde programas de código de máquina em fita e examine-os na memória. Antes de entrar o programa, é necessário dar o comando **CLEAR** apropriado. Feito isso, o programa é entrado e rodado; em seguida, o computador pede que você forneça um endereço inicial, que deverá ser o da memória, acima do estabelecido em **CLEAR**. Lembre-se de que o comando **CLEAR** especifica o último endereço de BASIC, e o seu programa de código de máquina deverá começar na locação acima dele.

Agora digite o monitor de código de máquina para o seu computador.

```
10 CLS
20 PRINT @193,"1: ENTRAR EM COD
IGO DE MAQUINA "
30 PRINT @257,"2: EXAMINAR MEMO
RIA "
40 PRINT @321,"3: GRAVAR BYTES
NA FITA "
50 A$=INKEY$:IF A$<"1" OR A$>"3
" THEN GOTO 50
60 CLS
70 ON VAL(A$) GOSUB 200,400,600
80 GOTO 10
200 INPUT "ENDERECO INICIAL ";S
A
210 LINE INPUT D$
220 IF D$="" THEN GOTO 210
230 IF LEFT$(D$,1)="#" TTHEN RE
TURN
240 S$=LEFT$(D$,2)
250 POKE SA,VAL("&H"+S$)
260 PRINT SA,LEFT$(D$,2)
270 D$=MID$(D$,3)
280 SA=SA+1: GOTO  220
400 INPUT"ENDERECO INICIAL ";SA
410 PRINT"SAIDA PARA IMPRESSORA
?(S/N)"
420 A$=INKEY$:IF A$="" THEN 420
430 P=0:IF A$="S" THEN P=-2
440 PRINT #P,SA;
450 FOR M=0 TO 7
460 PRINT #P," ";RIGHT$(" "+HEX
$(PEEK(SA+M)),2);
470 NEXT:PRINT#P
480 SA=SA+8
490 A$=INKEY$:IF A$="" THEN 490
500 IF A$=CHR$(13) THEN RETURN
510 GOTO 440
600 INPUT "ENDERECO INICIAL ";S
A
610 INPUT "NUMERO DE BYTES ";N
620 LINE INPUT "NOME DO ARQUIVO
";N$
630 CSAVEM N$,SA,SA+N,SA
640 RETURN
```

```
  5 POKE 23658,8
 10 PRINT  INVERSE 1;AT 5,6;"
MONITOR  TK 90 X "
 15 PRINT AT 8,1;"1: ENTRAR EM
 CODIGO DE MAQUINA"
 20 PRINT AT 10,1;"2: EXAMINAR
 MEMORIA"
 30 PRINT AT 12,1;"3: GRAVAR B
YTES EM FITA"
 70 LET A$=INKEY$: IF A$<"1"
 OR A$>"3" THEN  GOTO 70
 80 CLS : GOSUB 100+200*(VAL A
 $-1)
 90 RUN
100 INPUT "ENDERECO INICIAL? "
;SA
110 INPUT  LINE D$
120 IF D$="" THEN  GOTO 110
125 IF D$(1)="#" THEN  RETURN
127 IF LEN D$=1 THEN  GOTO 110
130 LET S$=D$(1 TO 2)
140 FOR N=1 TO 2
150 IF S$(N)>"9" THEN  LET S$(
N)=CHR$ (CODE S$(N)-7)
155 IF S$(N)>"?" OR S$(N)<"0"
THEN  PRINT  FLASH 1;"CARACTER
E INVALIDO": GOTO 110
160 NEXT N
170 POKE SA,(CODE S$(1)-48)*16
+CODE S$(2)-48
180 PRINT SA,D$( TO 2)
190 LET D$=D$(3 TO )
200 LET SA=SA+1
210 GOTO 120
300 INPUT "ENDERECO INICIAL? "
;SA
310 INPUT "IMPRESSORA (S/N)? "
; LINE P$
320 LET ST=2: IF P$="S" THEN
LET ST=3
330 PRINT #ST;SA;
340 FOR M=0 TO 7
350 LET H$="##"
360 LET H$(1)=CHR$ (INT (PEEK
(SA+M)/16)+48)
370 LET H$(2)=CHR$ (48+PEEK (S
A+M)-16*(CODE H$(1)-48))
380 FOR N=1 TO 2
390 IF H$(N)>"9" THEN  LET H$(
N)=CHR$ (CODE H$(N)+7)
400 NEXT N
410 PRINT #ST;TAB 7+3*M;H$;
420 NEXT M
440 LET SA=SA+8
445 PRINT #ST: POKE 23692,0
450 LET A$=INKEY$: IF A$=""
THEN  GOTO 450
460 IF A$=CHR$ 13 THEN  RETURN

470 GOTO 330
500 INPUT "ENDERECO INICIAL? "
;SA
510 INPUT "NUMERO DE BYTES? ";
N
520 INPUT "NOME DO ARQUIVO? ";
LINE N$
530 IF LEN N$<1 OR LEN N$>10
THEN  GOTO 520
540 SAVE N$CODE SA,N
550 RETURN
```

Antes de executar este programa, você deve digitar uma linha 1 contendo um **REM** seguido do número apropriado de caracteres, de modo a receber seu programa em código de máquina (um caractere por byte de programa).

```
 10 PRINT AT 4,9;"MONITOR TK85"
 15 PRINT AT 8,4;"1 - PROGRAMAR
 EM CODIGO"
 20 PRINT AT 10,4;"2 - EXAMINAR
 A MEMORIA"
 50 LET A$=INKEY$
 60 IF A$<"1" OR A$>"2" THEN
 GOTO 50
 70 CLS
 80 GOSUB 100+200*(VAL A$-1)
 85 CLS
 90 RUN
100 PRINT AT 21,0;"ENDERECO INI
CIAL?"
105 INPUT SA
110 INPUT D$
120 IF D$="" THEN GOTO 110
125 IF D$(1)="$" THEN RETURN
130 LET S$=D$(1 TO 2)
135 SCROLL
140 FOR N=1 TO 2
150 IF S$(N)<"0" OR S$(N)>"F"
THEN PRINT "CARACTER INVALIDO"
155 IF S$(N)<"0" OR S$(N)>"F"
THEN GOTO 110
160 NEXT N
170 POKE SA,(CODE S$(1)-28)*16+
CODE S$(2)-28
175 SCROLL
180 PRINT SA,D$( TO 2)
190 LET D$=D$(3 TO )
195 IF LEN D$=1 THEN GOTO 110
200 LET SA=SA+1
210 GOTO 120
300 PRINT AT 21,0;"ENDERECO INI
CIAL?"
310 INPUT SA
320 SCROLL
330 PRINT SA;
340 FOR M=0 TO 7
350 LET H$="  "
360 LET H$(1)=CHR$ (INT (PEEK (
SA+M)/16)+28)
370 LET H$(2)=CHR$ (28+PEEK (SA
```

```
+M)-16*(CODE H$(1)-28))
400 PRINT TAB 7+3*M;H$;
410 NEXT M
440 LET SA=SA+8
450 LET A$=INKEY$
460 IF A$="" THEN GOTO 450
470 IF A$=CHR$ 118 THEN RETURN
480 GOTO 320
```

```
10 CLS
20 LOCATE 8,7:PRINT "1 - Progra
mar em código"
30 LOCATE 8,9:PRINT "2 - Examin
ar memória"
40 LOCATE 8,11:PRINT "3 - Grava
r bytes na fita"
50 A$=INKEY$:IF A$="" THEN GOTO
 50
60 CLS
70 ON VAL(A$) GOSUB 200,400,600
80 GOTO 10
200 INPUT "Endereço inicial";SA
210 LINE INPUT D$
220 IF D$="" THEN GOTO 210
230 IF LEFT$(D$,1)="#" THEN RET
URN
240 S$=LEFT$(D$,2)
250 POKE SA,VAL(&H0+S$)
260 PRINT SA,LEFT$(D$,2)
270 D$=MID$(D$,3)
280 SA=SA+1:GOTO 220
400 INPUT "Endereço inicial";SA
410 PRINT "Saída para impressor
a ? (S/N)"
420 A$=INKEY$:IF A$="" THEN 420
430 IF A$="S" OR A$="s" THEN OP
EN "LPT:" FOR OUTPUT AS #1 ELSE
 OPEN "CRT:"FOR OUTPUT AS #1
440 PRINT #1,SA;
450 FOR M=0 TO 7
460 PRINT #1," ";RIGHT$(" "+HEX
$(PEEK(SA+M)),2);
470 NEXT:PRINT #1,
480 SA=SA+1
490 A$=INKEY$:IF A$="" THEN 490
500 IF A$=CHR$(13) THEN CLOSE #
1:RETURN
510 GOTO 440
600 INPUT "Endereço inicial ";S
A
610 INPUT "Número de bytes ";N
620 LINE INPUT "Nome do arquivo
 ";N$
630 X$="CAS:"+N$
640 BSAVE X$,SA,SA+N,SA
650 RETURN
```

COMO O MONITOR FUNCIONA

Quando você rodar o programa apropriado à sua máquina, ele exibirá um menu onde aparecem as três opções básicas de trabalho com o monitor: "Entrar Código de Máquina", "Examinar a Memória" ou — exceto no ZX-81 — "Armazenar Bytes na Fita".

Se você quiser entrar código de máquina, deverá pressionar a tecla 1. Depois de fornecer um endereço inicial para a máquina e pressionar <**ENTER**> ou <**RETURN**>, você deve começar a digitar os códigos de máquina na forma de valores hexadecimais de dois dígitos cada. Cada valor deve ser separado do seguinte por um espaço em branco.

Antes de pressionar <**ENTER**> ou <**RETURN**> você pode entrar tantos pares quantos desejar. Mas é melhor digitar uma linha por vez e checar rigorosamente os dígitos antes de pressionar <**ENTER**>. É muito mais fácil editá-los enquanto eles ainda estão na tela do que quando estiverem na memória.

Você notará que as versões do programa para o Spectrum e para o ZX-81 traduzem seus números hexa para decimais, antes de colocá-los na memória. Isto deve ser feito, como explicamos antes, porque o **POKE** é uma declaração do BASIC, e o BASIC nestas duas máquinas não aceita números hexa. Mesmo assim, é melhor você pensar apenas em hexa pois isso lhe dará uma melhor percepção de como o computador funciona.

Você deve finalizar os seus programas de código de máquina com um sinal sustenido # ou, no ZX-81, um sinal de dólar. Isso trará o menu de volta para a tela.

Se você selecionar a opção 2 para examinar a memória, o programa pedirá novamente um endereço inicial — desta vez é o início da área de memória que precisa ser observada. Para examinar todo o seu programa em código de máquina, esse endereço inicial deverá ser o mesmo que o endereço inicial que você forneceu anteriormente.

A seguir, exceto no ZX-81, será perguntado se você quer imprimir uma cópia definitiva do programa em código de máquina na sua impressora. Se você não quiser fazer isso, e pressionar N, o programa imprimirá o endereço inicial e os conteúdos dessa locação e das sete locações de memória subseqüentes em uma linha na tela.

Se, ao contrário, você responder à questão acima pressionando qualquer tecla do teclado — exceto <**ENTER**> ou <**RETURN**> —, o endereço original será impresso, junto com seus conteúdos e os conteúdos dos sete bytes subseqüentes. Assim, seguindo esse método, você poderá imprimir todo o seu programa de código de máquina.

Se você localizar um erro e quiser corrigi-lo, pressione <**ENTER**> ou <**RETURN**>, e o programa retornará ao menu. Pressione 1 para entrar o código de máquina, e o computador pedirá novamente um endereço inicial. Desta vez, forneça o endereço do byte que estiver errado. Se o erro atingir toda uma série, forneça apenas o endereço do primeiro byte da série; em seguida, entre os bytes corretos.

Outro modo de fazer correções, quando existe apenas um erro, consiste em acionar a tecla <**BREAK**> (nos micros da linha Sinclair e TRS-Color), ou as teclas <**CONTROL**> <**STOP**> (no MSX), para interromper o programa monitor, e usar o **POKE** de modo direto, para corrigir o conteúdo da locação apropriada. Lembre-se de que no Spectrum e no ZX-81 você deve usar o **POKE** com um número decimal. Assim que terminar a correção, pressione <**ENTER**> ou <**RETURN**> e examine mais uma vez a memória para ter certeza de que dessa vez você acertou.

GUARDE SUAS ROTINAS

Se você quiser conservar o monitor de código de máquina, deverá guardá-lo em uma fita separado de seus programas de código de máquina, da maneira usual. A opção "Armazenar Bytes em Fita" no menu do programa monitor serve apenas para guardar as rotinas de código de máquina que foram entradas com o monitor. A única exceção é para o ZX-81, no qual você deve guardar suas rotinas de código de máquina junto com o programa monitor.

Nas outras máquinas, se você pressionar 3, o programa pedirá mais uma vez um endereço inicial. Este deverá ser normalmente o endereço inicial da rotina de máquina que você acabou de entrar, embora qualquer parte da memória possa ser guardada por meio da utilização de um endereço diferente.

As versões deste programa para o Spectrum, o TRS-Color e o MSX pedirão então a você os números de bytes que a sua rotina em código de máquina ocupa. Você pode calcular isso pela contagem dos pares de dígitos hexas que foram entrados anteriormente. Cada par é um byte. Em seguida, você deve entrar um nome para a rotina — assim, o seu computador será capaz de identificá-lo quando você quiser carregá-lo. A rotina será gravada pressionando-se as teclas <**PLAY**> e <**RECORD**> no gravador, antes de acionar a tecla <**ENTER**> do computador. Quando a gravação se completar, aparecerá o menu na tela, novamente.

COMO CARREGAR CÓDIGO DE MÁQUINA

Com o Spectrum, o TRS-Color e o MSX a rotina de código de máquina é carregada de modo normal (por exemplo, no MSX deve-se utilizar o comando **BLOAD "CAS:ROTINA"**, onde

ROTINA é o nome dado à rotina quando esta foi gravada). Contudo, se o monitor na memória da máquina estiver rodando, você deverá antes disso pressionar <BREAK> ou <CONTROL> <STOP>, para interrompê-lo.

Lembre-se de repetir **CLEAR** para proteger uma área de memória para o programa em código de máquina, caso você tenha desligado a máquina, ou alterado a posição do RAMTOP desde que o digitou.

Com o ZX-81, o seu programa em código de máquina será automaticamente carregado da fita junto com o monitor.

ROLE PARA TRÁS

Para exemplificar o uso do monitor, tente o programa listado abaixo. Com exceção dos micros da linha MSX, esse programa rola a tela para trás — isto é, de cima para baixo e não o contrário. Para o MSX, a rotina apresentada serve para preencher a tela com traços coloridos (é o mesmo programa apresentado na primeira lição desta série). Entre a rotina adequada à sua máquina, utilizando o programa monitor. Termine digitando um sinal #.

```
8E 05 E0 A6 84 A7 89 FE
00 30 88 E0 A6 84 A7 88
20 8C 04 00 2C F3 30 89
02 01 8C 06 00 25 E4 39
#
```

```
21 9F 58 11 BF 58 06 08
25 15 E5 D5 C5 01 A0 00
ED B8 06 02 C5 D5 11 00
F9 19 D1 01 20 00 ED B8
E5 21 00 F9 19 EB E1 01
E0 00 EB B8 C1 10 E5 C1
D1 E1 10 D4 21 00 3F 06
08 C5 24 E5 AF 77 54 5D
13 01 1F 00 ED B0 E1 C1
10 EF 21 9F 5A 11 BF 5A
01 A0 02 ED B8 21 00 58
11 01 58 3A 8D 5C 77 01
1F 00 ED B0 C9 #
```

```
01 18 03 2A 0C 40 09 54
5D 01 F7 02 2A 0C 40 09
ED B8 2A 0C 40 23 36 00
54 5D 13 01 1F 00 ED B0
C9 $
```

```
98 ED 6B 02 03 3E 19 06
FF ED B3 3D 20 FA C9 02
18 F9 C9 #
```

RODE OS PROGRAMAS

Uma vez digitado o programa em código de máquina e verificados os possíveis erros, chegou o momento de rodá-lo. Lembre-se que ele não reagirá ao comando **RUN** normal. Para rodar programas em código de máquina, são necessárias instruções especiais que variam de computador para computador.

O Spectrum e o ZX-81 utilizam a função **USR** (abreviatura de USer Routine) do BASIC, seguida de um endereço inicial do programa em código de máquina. O **USR** faz retornar o valor decimal contido no registro interno BC do microprocessador, uma vez que se tenha completado a execução da rotina em código de máquina. Isto não é muito útil para você, no momento, mas significa que se esse retorno não ocorrer normalmente o programa em código de máquina não deve ter rodado corretamente.

O **USR** não é um comando, mas sim uma função. E a estrutura do BASIC do Sinclair exige que uma linha executada seja iniciada com um comando. Assim, o **USR** deve ser prefaciado por uma palavra de comando. Se você utilizou os endereços iniciais sugeridos neste artigo, para o Spectrum, faça:

```
RANDOMIZE USR 32000
```

e no ZX-81 utilize:

```
RAND USR 16514
```

(observe que a tecla **RAND** no Spectrum fornece **RANDOMIZE** na tela. No ZX-81 a tecla **RAMD** fornece **RAND** na tela). Em si próprio, o comando não tem sentido, mas ele dá resultado quando você rodar a rotina em código de máquina. Lembre-se de que a linha acima é apenas um exemplo. Troque o número que vem depois de **USR** pelo endereço decimal inicial de seu programa, fornecido para o monitor quando digitado.

Na verdade, qualquer comando BASIC pode ser utilizado para prefaciar **USR.** Por exemplo, o **PRINT USR 32000**, embora este, na verdade, imprima o valor do par de registros BC na tela, e isso sempre seja desejável.

RANDOMIZE USR 32000 ou **RAND USR 16514)** é mais utilizado porque evita efeitos colaterais indesejados. Mas deverá ser evitado se você estiver utilizando números aleatórios em qualquer parte do programa, pois ele irá acionar o gerador de números aleatórios novamente. Então, utilize no Spectrum:

```
LET L = USR 32000
```

e no ZX-81 empregue:

```
LET L = USR 16514
```

Isso coloca o valor dos registros BC na variável **L**, quando o programa em código de máquina termina de rodar — isso pode não ter nenhuma utilidade para o seu programa, mas no Sinclair os comandos **LET**, **L**, = e **USR** estão todos nas mesmas teclas, o que simplifica o trabalho de digitar.

O TRS-Color também possui instruções para executar programas em código de máquina. **EXEC** é um desses comandos; deve ser seguido pelo endereço inicial da rotina em código de máquina que você quiser rodar. Por exemplo:

```
EXEC 24000
```

rodará a rotina do código de máquina que se inicia na locação de memória em 24000 decimal. Existe também a função **USR**, que aceita parâmetros ou variáveis para serem passados entre o programa BASIC e a rotina em código de máquina. Antes de começar, você precisa definir onde deve se iniciar a rotina em código de máquina. Isso deve ser feito por meio da instrução **DEFUSR**, seguida de um número de 0 a 9. Assim, você pode definir até 10 rotinas **USR** diferentes dentro de um programa em BASIC. Quando **DEFUSR** não é seguido de um número o computador estabelece que seu significado equivale a **DEFUSR 0**. A sintaxe normal em um programa BASIC é:

```
10 DEFUSR1 = 24000
```

Isso define a rotina **USR** número 1, como se iniciando na locação 24000 decimal da memória.

A função que realmente executa o código de máquina é **USR.** Devido a um defeito na ROM, para chamar uma rotina de código de máquina do TRS-Color, **USR** deve ser seguida de 0 e o número da rotina que você já definiu.

A sintaxe normal é:

```
P = USR01(Q) ou PRINT USR01(Q)
```

A função **USR** copia o valor da variável **Q** no acumulador interno do microprocessador. Assim, ela pode ser captada e usada pela rotina em código de máquina, se for necessário.

Neste exemplo, quando a rotina em código de máquina termina sua execução, os conteúdos do acumulador retornam ao programa BASIC, como variável **P.**

Como em outros casos, variáveis de

string também podem ser transmitidas entre o programa em BASIC e programas em código de máquina.

Para executar programas em código de máquina, o MSX utiliza a função do BASIC chamada **USR** (abreviatura do USer Routine), seguida de um endereço inicial do programa de código de máquina. A **USR** retorna o valor decimal contido em um par de registros internos do microprocessador, uma vez que se tenha completado a execução da rotina em código de máquina. Isto não é muito útil para você, no momento, mas, se este retorno não ocorrer normalmente, significa que o programa em código de máquina não deve ter rodado corretamente.

A função **USR** aceita também parâmetros ou variáveis para serem passados entre o programa BASIC e a rotina em código de máquina.

Como já foi dito anteriormente, **USR** é uma função e não um comando. Ora, a estrutura do BASIC do MSX impõe que ela esteja contida numa linha iniciada com um comando. Podemos utilizar então algo do gênero:

```
P = USR(Q) ou PRINT USR(Q)
```

A função **USR** copia o valor da variável **Q** no acumulador interno do microprocessador. Assim, ele pode ser captado e usado pela rotina em código de máquina, em caso de necessidade.

Tal como nos exemplos já vistos, quando a rotina em código de máquina termina sua execução, os conteúdos do acumulador retornam ao programa BASIC, como a variável **P.** Para a rotina apresentada neste programa, é necessário preparar a tela gráfica antes de chamá-la, pois ela não faz isso internamente. Para executá-la, faça o seguinte programa e rode-o com o comando **RUN:**

```
1 SCREEN 2
2 PSET(0,0)
3 A=USR(B)
4 GOTO 4
```

Como foi reservado o espaço em RAM acima de &HE000, usamos o endereço inicial - 8191 ou §HE001, em hexadecimal, de modo a iniciar o **USR.** O número em decimal é negativo pois é o complemento do número decimal que normalmente apareceria como negativo.

Uma observação final para a rotina de exemplo para o **MSX**: ela usa a tela gráfica (**SCREEN 1**); por isso, se o leitor acidentalmente rodar a rotina sem o programa dado acima (que retorna as coisas ao normal), deverá digitar **SCREEN 0** e pressionar a tecla **<RETURN>** depois.

O QUE SÃO VARIÁVEIS

Todos aqueles Xs e Ys esquisitos nos programas tornam-se simples quando se entende realmente sua função. Aprenda como trabalhar com variáveis em seus próprios programas.

Quando você quiser armazenar uma informação qualquer na memória do computador, será necessário, identificá-la; do contrário, o computador será capaz de descobrir onde está a informação no momento em que você precisar dela novamente. Muitas vezes a informação que se pretende armazenar consiste em um número cujo valor se modifica sempre que o computador executa uma repetição (laço) em uma seção do programa. Abaixo temos um exemplo de uma situação desse tipo:

```
10 LET X=0
20 LET X=X+1
30 PRINT X;" ";
40 FOR T=0 TO 10
50 NEXT T
60 GOTO 20
```

A linha 60 cria o laço, fazendo com que o programa se repita indefinidamente. O número representado por X (originalmente zerado pela linha 10) atinge a linha 20 e é aumentado em 1. O X é chamado de variável numérica, pois armazena um número.

Para entender melhor como funciona uma variável numérica, imaginemos que a memória do computador é constituída por uma série de caixas ou compartimentos. Cada caixa recebe um nome. Num computador simples, esses nomes poderão ser letras do alfabeto. Para armazenar um número na memória, bastará atribuir esse valor a uma das caixas:

```
LET C = 25
```

Essa instrução armazena o valor 25 na variável C. Nem todos os computadores exigem o emprego do comando **LET**, mas é uma boa idéia incluí-lo até que você esteja experiente.

Uma vez armazenado, esse valor pode ser utilizado por outras declarações em outros cálculos, etc. Você pode digitar, por exemplo:

```
PRINT C * 4
```

ou:

```
PRINT C/5
```

Essas declarações deverão fornecer as respostas 100 e 5 respectivamente, se continuarmos com o valor definido para **C**, acima. Mas se você digitar:

```
PRINT C
```

verá que o valor de **C** permanece inalterado. Para modificar o valor de **C**, aumentando-o para 30, por exemplo, digite diretamente:

```
LET C = 30
```

ou, ainda:

```
LET C = C + 5
```

Note que a linha de comando acima não faz sentido do ponto de vista matemático. Entretanto, o comando **LET** não deve ser entendido como uma equação, apesar do sinal de igualdade. A declaração acima significa apenas: "tome o valor numérico armazenado na variável **C**, some 5 a ele, e coloque de volta o resultado na variável **C**". O que acontece é que o valor anterior presente em **C** (no caso, 25) é apagado quando armazenamos um novo valor na variável.

OS NOMES DAS VARIÁVEIS

A combinação de caracteres permitidos pelo BASIC para se dar nomes às variáveis numéricas difere de computador para computador. As principais regras são mostradas no box na página 99.

Mas uma coisa que nenhum computador aceitará é um nome de variável que se inicie por um número. Por exemplo:

```
LET 7 = 14
```

não tem sentido. Um nome de variável pode incluir números (exemplos: VAR7, A1985, etc.), mas o primeiro caractere precisa ser sempre uma letra. Além disso, não é permitido utilizar como nome de variáveis palavras que representem comandos, como **TO, THEN, OR** ou **AND**, etc. Alguns micros não aceitam que essas palavras-chaves façam parte do nome da variável: a variável BEND, não seria aceita pelo interpretador BASIC, pois inclui a expressão **END**.

VARIÁVEIS EM AÇÃO

A maioria dos programas que você escrever utilizará diversas variáveis. Eis

um programa curto que simula a ação de uma bomba "universal" de combustível (gasolina, álcool, etc.) e que funciona de maneira muito interessante:

```
10 LET LITROS=0
20 LET CRUZ=0
30 CLS:PRINT
40 PRINT " ESCOLHA O COMBUSTIVE
L:          aLCOOL      gASOLINA
   dIESEL "
50 INPUT A$
60 IF A$="A" THEN QTY=0.3
70 IF A$="G" THEN QTY=0.2
80 IF A$="D" THEN QTY=0.4
90 CLS
100 PRINT @130,"COMBUSTIVEL   ";
A$
```

- O QUE É UMA VARIÁVEL?
- COMO SÃO UTILIZADAS AS VARIÁVEIS?
- VARIÁVEIS DE CONTROLE PARA LAÇOS **FOR...NEXT**
- O QUE É CORDÃO (STRING)
- VARIÁVEIS NUMÉRICAS E ALFANUMÉRICAS
- CORDÕES VAZIOS E PROGRAMAÇÃO DE JOGOS

```
110 PRINT @199,"LITROS"
120 PRINT @256,"PRECO A PAGAR"
130 LET K=PEEK(344)
140 IF K=251 THEN CLS:GOTO 10
150 IF K=253 THEN LET CRUZ=CRUZ
+1:LET LITROS=INT((LITROS+QTY)*
10)/10
160 PRINT @206,LITROS
170 PRINT @270,CRUZ;"MIL CRUZEI
ROS"
200 GOTO 130
```

Atenção: o programa listado acima rodará apenas nos compatíveis com o TRS-Color. Para poder executá-lo também em micros da linha TRS-80, faça as seguintes modificações:

```
100 PRINT @258,"COMBUSTIVEL
    ";A$
110 PRINT @385,"LITROS"
120 PRINT @512,"PRECO A PAGAR"
160 PRINT @396,LITROS;
170 PRINT @526,CRUZ;"MIL CRU-
    ZEIROS";
```

```
10 LET litros=0
20 LET cruz=0
40 PRINT " Escolha o combusti
vel: "
45 PRINT "A(lcool)   G(asolin
a)   D(iesel)"
50 INPUT A$
60 IF A$="a" THEN  LET qty=1/
3
70 IF A$="g" THEN  LET qty=1
5
80 IF A$="d" THEN  LET qty=1/
3
 90 CLS
100 IF INKEY$="n" THEN  LET cr
uz=cruz+1: LET litros=INT ((li
tros+qty)*10)/10
105 PRINT AT 6,2;"COMBUSTIVEL
 ";A$
110  PRINT AT 9,7;"LITROS"
120 PRINT AT 9,15;litros;
130 PRINT AT 12,0;"PRECO A PAG
AR"
140 PRINT AT 12,15;cruz;" mil
cruzeiros"
150 IF INKEY$="f" THEN  CLS :
GOTO 10
200 GOTO 100
```

```
 5 CLS
10 LET LITROS=0
```

```
20 LET CRUZ=0
40 PRINT " ESCOLHA O COMBUSTIV
EL: "
45 PRINT  "A(LCOOL)   G(ASOLIN
A)    D(IESEL)"
50 INPUT A$
60 IF A$="A" THEN LET QTY=1/3
70 IF A$="G" THEN LET QTY=1/5
80 IF A$="D" THEN LET QTY=1/3
90 CLS
100 IF INKEY$<>"N" THEN GOTO 10
5
101 LET CRUZ=CRUZ+1
102 LET LITROS=LITROS+QTY
105 PRINT AT 6,2;"COMBUSTIVEL
";A$
110 PRINT AT 9,7;"LITROS"
120 PRINT AT 9,15;LITROS;
130 PRINT AT 12,0;"PRECO A PAGA
R"
140 PRINT AT 12,15;CRUZ;" MIL C
RUZEIROS"
150 IF INKEY$="F" THEN RUN
200 GOTO 100
```

```
5 REM ***TECLE A BARRA DE ESPAÇ
OS PARA LIGAR A BOMBA E D PARA
DESLIGÁ-LA***
10 LET LITROS=0
20 LET CRUZEIROS=0
30 CLS:LOCATE 10:PRINT"BOMBA DE
 COMBUSTÍVEL"
40 PRINT:PRINT:PRINT"Escolha o
seu combustível"
50 LOCATE 10,5:PRINT"1- alcool"
60 LOCATE 10,6:PRINT"2- gasolin
a"
70 LOCATE 10,7:PRINT"3- diesel"
80 LOCATE 15,12:INPUT"Sua opção
: ";C
90 IF C=1 THEN P=1/3.1
100 IF C=2 THEN P=1/4.7
110 IF C=3 THEN P=1/3.5
120 CLS
130 LOCATE 10,5:PRINT"Combustív
el ";C
140 LOCATE 10,10:PRINT"Preço a
pagar =>"
150 LOCATE 17,12:PRINT"Litros=>
"
160 IN$=INKEY$:IF IN$=CHR$(68)
THEN END
165 IF IN$<>CHR$(32)THEN 160
170 CR=CR+1000:LI=LI+P
180 LOCATE 26,10:PRINT CR
190 LOCATE 26,12:PRINT LI
200 GOTO 160
```

```
5  REM  **TECLE A BARRA DE ESPA
COS PARA RODAR A BOMBA E F PARA
 TERMINAR
10  LET LITROS = 0
20  LET CRUZEIROS = 0
30  HOME : HTAB 10: PRINT "BOMB
A DE COMBUSTIVEL"
40  PRINT : PRINT "Escolha o se
u combustivel:"
```



```
50  PRINT : HTAB 5: PRINT "1- A
LCOOL"
60  HTAB 5: PRINT "2- GASOLINA"
70  HTAB 5: PRINT "3- DIESEL"
80  VTAB 15: HTAB 10: INPUT "Op
cao =>";C
90  IF C = 1 THEN P = 1 / 3.1
100  IF C = 2 THEN P = 1 / 4.7
110  IF C = 3 THEN P = 1 / 3.5
120  HOME
130  VTAB 5: HTAB 10: PRINT "Co
mbustivel ";C
140  VTAB 10: HTAB 10: PRINT "P
reco a pagar =>"
150  PRINT : HTAB 17: PRINT "Li
tros =>"
160  GET IN$: IF IN$ =  CHR$ (7
0) THEN  END
165  IF IN$ <  >  CHR$ (32) THE
N 160
170 CR = CR + 1000:LI = LI + P
180  VTAB 10: HTAB 26: PRINT CR
190  VTAB 12: HTAB 26: PRINT  I
NT (LI * 100 + .5) / 100
200  GOTO 160
```

Vale a pena experimentar esse programa de muitas maneiras, até entender como ele funciona, pois há nele um conjunto razoável de variáveis.

A função de algumas delas é bastante evidente: a variável A, por exemplo, seleciona o tipo de combustível, através de um número (1, 2 ou 3) e determina, nas linhas 60 a 80, qual é a fração de litro do combustível escolhido que vale mil cruzados.

A tecla N aciona a bomba que mede o fluxo de combustível. O programa começa então a incrementar o número de litros de combustível (variável **CRUZ**, inicialmente zerada na linha 20, bem como a variável **LITROS**). Enquanto a tecla N estiver sendo pressionada, o programa continuará avançando; ao atingir a linha 200, ele retornará à linha 130 (100 no Spectrum), e manterá o medidor funcionando. Para terminar de abastecer, pressione a letra F.

A presença na tela de frações decimais com muitos dígitos revela a imprecisão própria do computador, quando é feita a conversão de números binários internos, para decimais comuns.

VARIÁVEIS DE CONTROLE

Assim como são utilizadas em declarações **LET**, as variáveis também podem aparecer em laços **FOR...NEXT**, como você já deve ter percebido. Neste caso, a variável que é alterada a cada repetição do laço é chamada de *variável de controle*.

Em alguns computadores, os caracteres empregados para dar nome às variáveis de controle não podem ser aplicados a variáveis numéricas (veja quadro). Nos micros da linha Sinclair, por exemplo, você pode digitar:

```
LET SCORE = 0
LET RECORDE = 0
```

para variáveis numéricas, mas não pode utilizar nomes como estes em instruções **FOR...NEXT**, tais como em:

```
FOR TEMPO = 1 TO 99999
```

Ao invés disso, você é obrigado a empregar a variável de controle com uma única letra, como por exemplo T. Os outros computadores (linhas TRS, Apple e MSX) não têm essa exigência.

VARIÁVEIS ALFANUMÉRICAS

E se, além de números, quisermos armazenar na memória do computador informações como nomes, endereços, etc.?

As variáveis capazes de armazenar um *conjunto de caracteres,* incluindo letras e números, são chamadas de variáveis alfanuméricas, em BASIC. Outra denominação usada é a de cordão (ou *string,* em inglês). Para facilitar a compreensão desse novo elemento, podemos partir de uma comparação: um cordão funciona como uma espécie de sacola de compras. Você pode enchê-la com um grande número de artigos e manejá-la como uma unidade — sem ter que ocupar-se com cada artigo separadamente. Da mesma forma, um único nome de variável alfanumérica serve para caracterizar várias memórias, simultaneamente.

Os cordões podem conter quase tudo: letras, números, sinais de pontuação, espaço em branco, e até mesmo caracteres gráficos. Eles devem estar sempre delimitados entre aspas

"NOMES E ENDEREÇOS DE CLIENTES".

"POR FAVOR ENTRE AGORA SUA RESPOSTA"

"24 de Janeiro"

"01-7346710"

"NE 135 S 181"

O exemplo seguinte também é um cordão, embora apenas alguns computadores, como o Spectrum, o TK-2000 e o MSX, permitam que se possa digitá-los desta forma diretamente no teclado:
"   "

A linguagem BASIC tem recursos para manipular cordões alfanuméricos de diversas maneiras. Por exemplo, existem funções e operações para juntar dois ou mais cordões uns nos outros (operação de concatenação), ou para dividi-los em dois ou mais subcordões.

Entretanto, o computador não "entende" o que está dentro de um cordão e reproduz o conteúdo deste sempre de forma exatamente igual, mesmo que ele esteja escrito em sânscrito ou alemão.

Não é possível, também, efetuar operações matemáticas com um cordão, como somas, multiplicações, divisões e outras.

Para provar isso, rode o programa abaixo:

```
10 PRINT "2+2= ";
20 FOR N=1 TO 40
25 NEXT N
30 PRINT 1+2*2
```

O conteúdo do cordão (entre aspas) é impresso como você o entrou. Na verdade, somente a linha 30 é calculada, e seu resultado, impresso (a linha 20 simula o tempo que o computador está "gastando" para pensar, antes de fornecer sua resposta). No MSX, no Apple e no TRS-80, modifique 40 para 2000, para um retardo razoável de tempo.

Mas se quiser utilizar o mesmo cordão mais de uma vez, pode economizar tempo de digitação (e espaço em memória) criando um rótulo para ele. Esse rótulo é chamado também de variável alfanumérica. Sua extensão pode variar de um computador para outro (veja quadro) mas a variável deve ser sempre seguida por um sinal de cifrão ($).

Eis aqui, por exemplo, um sistema automático de pedidos em uma lanchonete:

```
10 LET A$="POR FAVOR,DIGITE QU
ANTOS"
20 LET B$=" VOCE QUER"
30 PRINT A$;"HAMBURGERES";B$
35 INPUT H
40 PRINT A$;"SACOS DE FRITAS";
B$
45 INPUT S
50 PRINT A$;" MILK SHAKES";B$
55 INPUT M
60 PRINT "OBRIGADO. IRA CUSTAR
 ";(H*  15000+S*5000+M*13000);
" POR FAVOR."
```

A possibilidade de reduzir o tamanho dos programas é a razão pela qual sistemas de processamento de dados e de textos — que incluem informações e mensagens muito utilizadas, tais como "número da conta do cliente" e "preço por milhar" e "18% de ICM" — utilizam variáveis alfanuméricas.

Você também poderá empregá-las na programação de jogos. Suponhamos que, em um jogo, seja necessário a todo momento colocar na tela uma linha enorme de símbolos gráficos — o muro de uma prisão para um jogo de aventuras, por exemplo. Tudo o que você precisa fazer é digitá-lo uma única vez, "rotulá-lo" e

Variáveis: o que você pode e não pode usar

Tipo de variável				
Variável numérica	Comprimento: sem limite, todas as letras são reconhecidas.	Comprimento: máximo de 255 caracteres, mas só as duas primeiras letras são reconhecidas.	Comprimento: máximo de 255 caracteres, mas só as duas primeiras letras são reconhecidas.	Comprimento: máximo de 255 caracteres, mas só as duas primeiras letras são reconhecidas.
	No ZX-81, só letras maiúsculas. No Spectrum maiúsculas e minúsculas, mas só maiúsculas são reconhecidas.	No TRS-Color, só letras maiúsculas. No TRS-80 há conversão automática para maiúsculas.	Conversão automática para maiúsculas.	Só letras maiúsculas: não há conversão automática.
	Caracteres especiais no nome não permitidos.	Caracteres especiais no nome não permitidos.	Caracteres especiais no nome não permitidos.	Caracteres especiais no nome não permitidos.
	Espaços no nome: permitidos mas não contam	Espaços no nome: não permitidos.	Espaços no nome: não permitidos.	Espaços no nome: não permitidos.
Variável de controle p/ **FOR...NEXT**	Uma letra apenas.	Variável numérica inteira ou de precisão simples.	Variável numérica sem restrições.	Variável numérica sem restrições.
Variável alfanumérica	Uma letra apenas, seguida de $.	Como nas variáveis numéricas, seguida de $, ou declarada pela inicial em **DEFSTR**.	Como nas variáveis numéricas, seguida de $, ou declarada pela inicial em **DEFSTR**.	Como nas variáveis numéricas, seguida de $.

utilizá-lo sempre que quiser.

Movimente esse cordão por toda a tela, como mostramos no programa:

```
5 CLS
10 LET B$="
                              "
20 LET A$=B$+"  UMA FELIZ PASCO
A PARA VOCES"+B$
30 FOR N=1 TO 65
40 PRINT @160,MID$(A$,N,32)
50 PRINT @320,MID$(A$,66-N,32)
60 SOUND N*3,1:NEXT N
80 GOTO 30
```

Para rodar o programa acima nos micros da linha TRS-80, substitua as linhas 40, 50 e 60; assim:

```
40 PRINT @320,MID$(A$,N,32)
50 PRINT @640,MID$(A$,66-N,32)
60 NEXT N
```

```
10 LET B$="
                              "
20 LET A$=B$+"    UMA FELIZ
PASCOA P  ARA VOCES   "+B$
30 FOR N=1 TO 65
40 PRINT  INK 2;AT 8,0;A$(N
TO N+31)
50 PRINT  INK 2;AT 12,0;A$(66
-N TO 97-N)
60 SOUND .02,N/2
70 NEXT N
80 GOTO 30
```

```
10 LET B$="
                              "
20 LET A$=B$+"    UMA FELIZ PA
SCOA PARA VOCES  "+B$
30 FOR N=1 TO 65
40 PRINT AT 8,0;A$(N TO N+31)
50 PRINT AT 12,0;A$(66-N TO 97
-N)
70 NEXT N
80 GOTO 30
```

```
10 CLS:CLEAR 1000
20 B$=SPACE$(40)
30 A$=B$+"        UMA FELIZ PASCO
A PARA VOCES        "+B$
40 FOR N=1 TO 80
50 LOCATE 0,7:PRINT MID$(A$,N,3
9)
60 LOCATE 0,14:PRINT MID$(A$,81
-N,39)
65 FOR X=1 TO 50 : NEXT
70 NEXT
```

```
10  HOME
20 B$ = "
                         ": REM  40 ES
PACOS
30 A$ = B$ + "        UMA FELIZ
PASCOA PARA VOCES         " + B$
35  INVERSE
40  FOR N = 1 TO 80
50  VTAB 7: PRINT  MID$ (A$,N,4
0)
60  VTAB 14: PRINT  MID$ (A$,81
 - N,40)
65  FOR X = 1 TO 100: NEXT
70  NEXT
80  GOTO 40
```

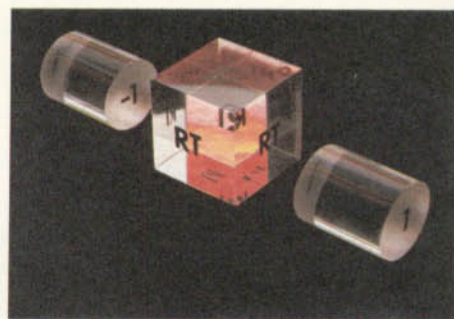
Aqui você tem um cordão enorme, como mostramos na linha 20 — todos os espaços na linha 10, mais a mensagem, colocados juntos. Mais adiante, veremos como isso é feito, e como o cordão é "cortado" novamente para caber na tela. Mas, se nesse meio tempo você quiser modificar a mensagem, certifique-se de que sua nova declaração, incluindo os espaços, tem a mesma extensão da original. De outro modo você terá que descobrir o que significam os números nas linhas 40 a 55 e modificá-los.

As variáveis alfanuméricas são também muito empregadas quando o programa precisa entrar uma informação que é sempre modificada. Por exemplo:

```
10 PRINT "QUAL E O SEU NOME "
20 INPUT N$
30 PRINT "ALO, ";N$
```

Neste programa a linha 10 imprime o seguinte cordão na tecla: QUAL É O SEU NOME, seguido por um ponto de interrogação. O computador espera que você entre um nome (um outro cordão) e o chama de N$. Na linha 30, ele imprime o cordão "ALO", e chama de volta o cordão N$ armazenado na linha acima. Assim será exibido na tela:

```
ALO, TOM ( ou PAULO ou PEDRO )
```

ou qualquer outro. Observe que, aqui, os cordões QUAL É O SEU NOME e ALO não são armazenados dentro de variáveis, tais como **A$** ou **QUE$** ou **ALO$**, porque eles aparecem apenas uma vez no programa, e o computador não precisa se lembrar do que seja ALO — quando ele atinge a linha 20 simplesmente lê esse cordão e o imprime.

Ao mesmo tempo, é necessário dar um rótulo para o nome que vai ser entrado. Assim, ele poderá ser diferente cada vez que o programa for rodado. Ele é armazenado em uma variável, na linha 20 do programa, e usado na linha 30.

Existem computadores que admitem uma forma mais curta e mais clara de programação da declaração **INPUT**, através da combinação do **PRINT** "mensagem" da linha 10, com o **INPUT** da linha 20:

```
10 INPUT "QUAL E SEU NOME?" N$
20 PRINT"OLA, ";N$
```

CORDÕES VAZIOS

Outro modo de trabalhar com variáveis alfanuméricas já foi mostrado anteriormente na lição "Apontar...Fogo!" (págs. 28 e 33). Embora varie de computador para computador a forma de escrever uma variável, nesse caso, é esta:

```
20 IF A$ = "" THEN GOTO 10
```

As duas aspas sem nada entre elas são conhecidas como um cordão nulo ou vazio. A linha de programa significa: "se a entrada for igual a nada" — isto é, se nenhuma tecla estiver sendo pressionada — "volte para a linha 10 e espere até que uma tecla seja pressionada". Isto evita que o computador salte para outra parte do programa tão rapidamente que o jogador não tenha tempo de pressionar uma tecla de comando qualquer, ou de ver um resultado ou gráfico na tela.

Duas aspas com um espaço em branco entre elas, no entanto, são uma coisa muito diferente. Se você modificar a linha:

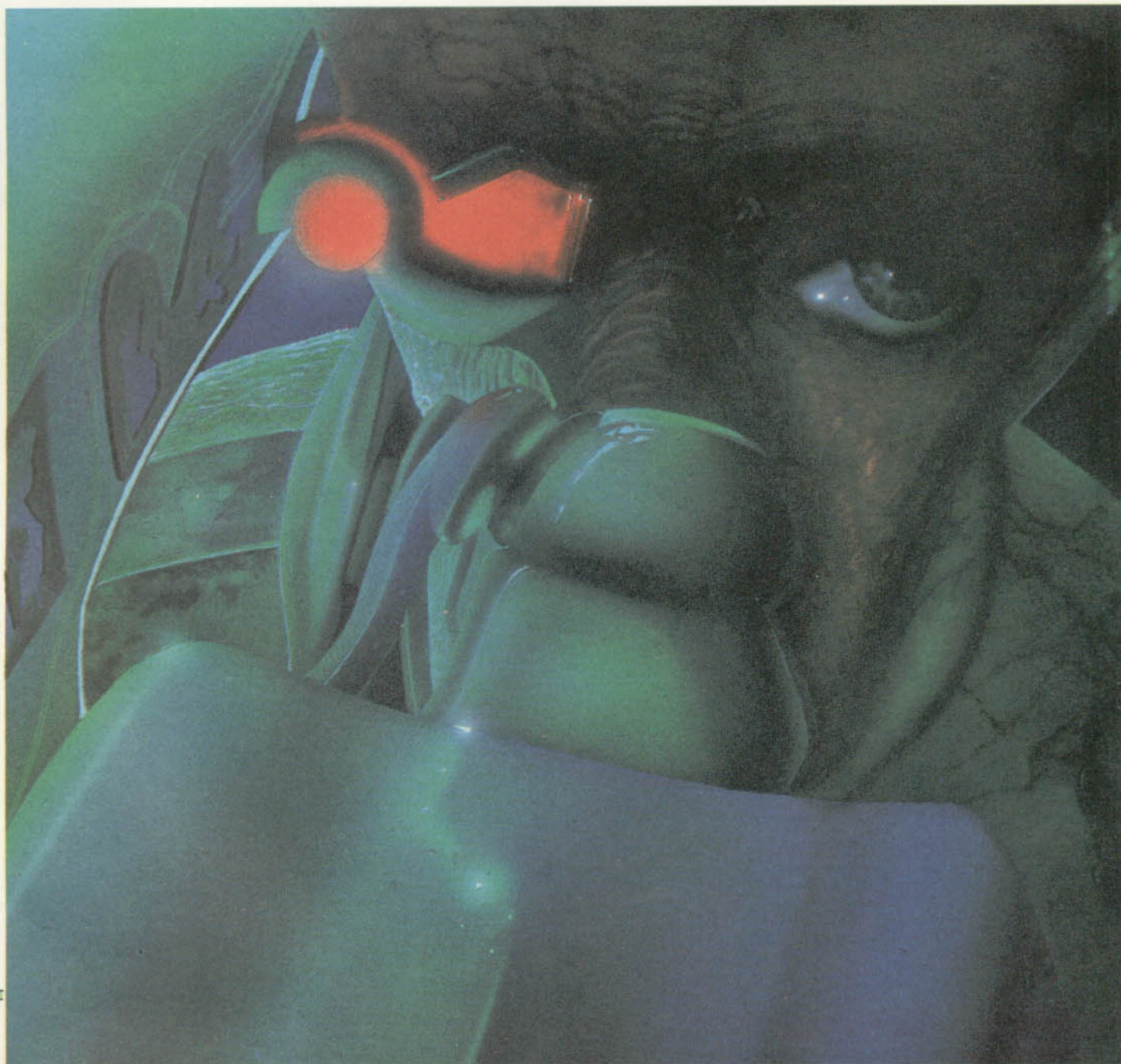
```
20 IF A$ = "  " THEN GOTO 10
```

o programa retrocederá à linha 10 apenas se o caractere entrado pelo jogador for um espaço — isto é, se ele estiver pressionando a tecla de espaço.

ATAQUE EXTRATERRESTRE

Batalhas interplanetárias foram sempre um dos pratos fortes dos livros de ficção científica, inspirando até mesmo a estratégia de "Guerra nas Estrelas" proposta por Ronald Reagan. No computador, elas podem ser tema de um jogo fascinante.

Os jogos para micros ficam muito melhores quando os recursos gráficos de alta resolução do computador são bem aproveitados. Esses gráficos permitem cenas muito mais detalhadas do que os simples caracteres ASCII (ou gráficos de baixa resolução), com os quais aprendemos a lidar até agora. Evidentemente, os programas que utilizam gráficos de alta resolução são bem mais complexos do que aqueles que trabalham apenas com os caracteres pré-programados do teclado, mas os resultados certamente valerão o esforço.

Muitos jogos para computador do tipo videogame envolvem oponentes, invasores extraterrestres ou outros adversários que revidam os disparos ao invés

de esperarem tranqüilamente que você os aniquile.

Agora, vamos ver um jogo chamado *Estação Espacial,* adequado a todos os tipos de computador (com exceção do ZX-81 e do Apple II), que vai ensiná-lo a fazer rotinas gráficas para movimentar aleatoriamente uma nave extraterrestre inimiga na tela, bem como lançar mísseis contra um alvo — no caso, a estação espacial controlada por você.

O jogador dispõe de 4 blindagens que poderá utilizar para se proteger dos ataques dos mísseis inimigos. Entretanto, você não pode mantê-las "ligadas" durante todo o tempo, pois a quantidade de combustível destinada a alimentar sua potência é limitada.

Para tornar o jogo mais difícil, o programa não só movimenta a nave inimiga de um lado para o outro, mas pode também fazê-la desaparecer subitamente dentro do hiperespaço e reaparecer em algum lugar inteiramente inesperado.

Do modo como foi descrito, o jogo não está realmente completo; assim, nada existe nele para medir a passagem do tempo e os pontos, o que pode diminuir o interesse pelo combate. Mas isso pode ser facilmente resolvido utilizando-se os métodos explicados nas páginas 61 a 67.

Embora a nave extraterrestre seja bem parecida com as dos fliperamas, a estação espacial é apenas um esboço. Você pode replanejá-la, se quiser gastar meia hora de modo interessante, utilizando para isso gráficos que você mesmo definirá. Os proprietários de micros da linha MSX podem ainda trabalhar com gráficos programáveis chamados *sprites,* que explicaremos em detalhes mais adiante. No entanto, mantenha o gráfico da sua nova estação espacial dentro da área utilizada por aquela definida neste jogo. Caso contrário, as blindagens defensivas poderão ser superpostas, tornando necessárias modificações mais drásticas do programa.

T

Digite e execute o programa abaixo:

```
10 PCLEAR 4:PMODE 4,1:PCLS
15 SCREEN 1,1
20 DIM AL(6),BL(6),BO(4)
30 DEFFNZ(X)=SGN(X)*SQR(V*V*X*X
/((127-AX)*(127-AX)+(95-AY)*(95
-AY)))
40 LET PW=250
50 FOR I=0 TO 7:READ A:POKE 153
6+I*32,A:NEXT I
60 GET(0,0)-(7,7),AL,G
65 GOTO 65
250 DATA 24,126,90,126,126,195,
129,129
```

Você verá a nave espacial extraterrestre aparecer na tela.

Os gráficos de alta resolução são apresentados na linha 10. Neste estágio a tela é ligada na linha 15; assim, você poderá ver como o programa funciona, porém, mais tarde, quando o jogo estiver completo, a linha será removida.

Os conjuntos que conterão a nave extraterrestre (AL), o míssil (BO) e um bloco em branco (BL) são dimensionados pela linha 20. A linha 30 utiliza a expressão **DEFFN** que significa *DEFine a FuNction.* Caso seja necessário empregar com freqüência uma longa expressão matemática no programa, a função predefinida o poupará de digitá-la diversas vezes em diferentes pontos do programa. Ela funciona, portanto, como uma espécie de sub-rotina em uma linha só. A expressão matemática na linha 30 é chamada de **FNZ** e será utilizada mais tarde no programa para movimentar um míssil em sentido diagonal, no vídeo. A linha 40 estabelece na tela a posição final de um indicador de combustível.

A linha 50 desenha a nave na tela, lendo os códigos gráficos correspondentes através de instruções **READ** e **DATA** na linha 250, e colocando-os na página de vídeo, no canto superior esquerdo da tela, com comandos **POKE.** A linha 60 usa a função **GET** para capturar esse padrão gráfico definido pelo bloco que vai de 0,7 a 7,7 na tela, e armazená-lo no conjunto denominado AL.

A linha 65 também é temporária. Tudo o que ela faz é manter a tela "congelada", de modo que se possa ver o resultado do processamento feito pelas linhas anteriores. Ela será removida mais tarde, quando o resto do jogo for acrescentado ao programa.

DESENHE OS MÍSSEIS

Agora, remova a linha 65, digitando o número 65 seguido de **<ENTER>**: depois, acrescente essas linhas e rode o programa.

```
70 FOR J=0 TO 4:READ A:POKE 153
6+J*32,A:NEXT J
80 GET (0,0)-(4,4),BO,G
85 GOTO 85
260 DATA 32,112,248,112,32
```

O míssil que será lançado pelo E.T. é lido em conjunto de códigos em **DATA,** e desenhado com o auxílio de comandos **POKE** no canto superior esquerdo da tela. Da mesma forma que anteriormente, usamos **GET** para registrar essa forma no conjunto BO.

Não importa que o foguete desenhado na tela se sobreponha ao desenho anterior da nave inimiga, deixando ainda, aparentemente, vestígios dela, pois a linha 80 apenas memoriza com o **GET** a área ocupada pelo míssil e não os seus arredores.

CONSTRUA A ESTAÇÃO ESPACIAL

Remova a linha 85, digitando 85 seguido de <**ENTER**>, acrescente essas linhas, e depois execute o programa.

```
90 LET AX=RND(248)-1:LET AY=RND
(178)+5
100 PCLS
110 CIRCLE(127,95),12,5:DRAW"BM
127,95;C5S48NUNLNDNR"
115 GOTO 115
```

A linha 90 escolhe uma posição aleatória para a nave E.T. A linha 100 limpa a tela e a linha 110 desenha a estação espacial, que é simplesmente um círculo (comando **CIRCLE**). O comando **DRAW**, no final dessa linha, traça uma cruz bem no meio da estação espacial. Esse comando (**DRAW**) pode ser imaginado como uma sucessão de declarações **LINE**, como já se explicou na lição anterior de programação BASIC. Ele será melhor explicado numa lição posterior.

DESENHE O INDICADOR DE COMBUSTÍVEL

Remova a linha 115 do mesmo modo que removeu as linhas 65 e 85. Acrescente essas linhas e aparecerão na tela maiores detalhes gráficos.

```
120 DRAW"BM131,87;S4D5BD6BLR3D2
L3D2R3BL12R3U2NL3U2NL3BU6U5G4R3
"
130 DRAW"BM5,1;L4D2NR4D2BE4BR2D
4R3U4BR5L3D2NR3D2R3BE4D4R3"
140 LINE(25,1)-(PW,5),PSET,BF
145 GOTO 145
```

A linha 120 traça na estação espacial um número que corresponde à quantidade de blindagens. Na linha 130 mostramos a palavra FUEL (combustível). Infelizmente, o TRS-Color não consegue escrever caracteres normais numa tela gráfica de alta resolução; por isso, letras e números devem ser literalmente "desenhados" com comandos **DRAW**.

A linha 140 exibe o indicador cheio de combustível para as blindagens, utilizando um método rápido para desenhar retângulos. Usamos a instrução **LINE** para traçar uma linha no alto da tela, indo da posição extrema esquerda (coordenadas 25,1) até a posição do combustível no momento (assinalada pela variável **PW**). O comando **PSET** diz ao TRS-Color para traçar essa linha em branco opaco e, deste modo, estabelece a cor. BF é uma abreviação para *box fill (preencher a caixa)* e "pinta" o retângulo com a cor utilizada pela linha original. Se você quiser traçar um retângulo vazio, utilize B ao invés de BF. Estão agora completos os gráficos de alta resolução para o jogo. O resto do programa está relacionado com o movimento do E.T. e do míssil e a ativação das blindagens.

MOVIMENTE A NAVE INIMIGA

Existem três sub-rotinas que devem ser acrescentadas ao programa. A primeira diz respeito ao movimento da nave inimiga. Digite-a, mas não a rode ainda porque não acontecerá nada.

```
1000 LET LX=AX:LET LY=AY
1010 IF RND(10)=1 THEN LET AX=R
ND(248)-1:LET AY=RND(178)+5
1020 LET AX=AX+RND(15)-8:LET AY
=AY+RND(15)-8
1030 IF AX>103 AND AX<144 AND A
Y>71 AND AY<112 THEN LET AX=LX:
LET AY=LY
1040 IF AX<0 THEN LET AX=-AX
1050 IF AX>248 THEN LET AX=497-
AX
1060 IF AY<6 THEN LET AY=12-AY
1070 IF AY>184 THEN LET AY=369-
AY
1080 PUT(LX,LY)-(LX+7,LY+7),BL,
PSET
1090 PUT(AX,AY)-(AX+7,AY+7),AL,
PSET
1100 RETURN
```

A nave E.T. (extraterrestre) é controlada de forma semelhante à base de mísseis da lição *Apontar...Fogo!* (páginas 28 a 33). A linha 1000 iguala as coordenadas da última posição da nave às coordenadas da posição atual, antes que ela seja movimentada.

A linha 1010 escolhe um número aleatório para que a nave inimiga possa mudar de posição na tela. Se esse número for 1 (ou seja, em 10% das vezes, em média), então o E.T. saltará para uma nova posição na tela. Se o número não for 1, será escolhida uma nova posição para a nave, distante entre -7 e +7 pontos na direção X (da esquerda para a direita), e entre -7 e +7 pontos na tela, na direção Y (de cima para baixo) — veja a linha 1020. A linha 1030 impede que a nave E.T. sobreponha-se à estação espacial, enquanto as linhas 1040 a 1070 não permitem que ela saia para fora da tela.

A nave invasora é apagada na linha 1080, colocando-se uma série de gráficos em branco nas últimas posições; em seguida, ela é posta na nova posição pela linha 1090. A linha 1100 retorna o programa à linha 160, que, a propósito, ainda não foi inserida.

COMO DISPARAR OS MÍSSEIS

A próxima sub-rotina serve para decidir, em primeiro lugar, se o míssil deve ser disparado ou não. Caso ocorra o disparo, ela estabelece a posição do míssil na tela e define com que blindagem a defesa vai bloqueá-lo.

```
2000 IF RND(7)<>1 THEN RETURN
2010 V=RND(8)+5:DX=FNZ(127-AX):
DY=FNZ(95-AY)
2020 IF DX<=0 AND DY>=0 THEN LE
T MA=1:GOTO 2050
2030 IF DX<=0 AND DY<=0 THEN LE
T MA=2:GOTO 2050
2040 IF DX>=0 AND DY<=0 THEN LE
T MA=3 ELSE LET MA=4
2050 LET MX=AX:LET MY=AY
2060 PUT(MX,MY)-(MX+4,MY+4),BO,
OR
2070 LET AF=1:RETURN
```

```
220 NEXT
230 LINE(PW,1)-(PW+2,5),PRESET,
BF
240 GOTO 160
```

Antes de rodar o programa, remova as linhas 15 e 145. Feito isso, você não verá o que está se desenvolvendo na tela durante a definição dos blocos gráficos. Rodado o programa ocorrerá uma curta pausa, antes que uma exposição completa apareça na tela.

A tela gráfica é agora ligada pela linha 150. A linha 160 checa se um míssil foi disparado. Se **AF = 0,** isso quer dizer que não houve disparos, e o programa salta para a sub-rotina que se refere ao lançamento do míssil, que começa na linha 2000; ou então o programa pula para a sub-rotina que movimenta o míssil; esta última começa na linha 3000. Depois, a nave E.T. movimenta-se. A linha 170 faz com que o programa salte para a sub-rotina na linha 1000.

A parte do programa das linhas 180 a 220 diz respeito à ativação das blindagens. A linha 190 verifica que tecla está sendo pressionada. Se a tecla for um número de 1 a 4, a linha 2000 desenhará a blindagem; se alguma das blindagens for acionada, a linha 210 diminuirá o combustível.

A linha 230 traça um retângulo negro no final do indicador do combustível, dando a impressão de que o suprimento de combustível está acabando, à medida que o valor em PW vai diminuindo. Finalmente, a linha 240 repete tudo de novo.

A linha 2000 decide se dispara um míssil. Existem seis chances em uma de que ele seja disparado, mas, se já há um míssil na tela, o programa não pode disparar outro. Se um foguete não puder ser disparado, a sub-rotina terminará.

A linha 2010 estabelece a extensão do deslocamento do míssil na tela — você pode pensar em **V** como se fosse a sua velocidade. O conteúdo de **V** é passado para a função **FNZ** — definida na linha 30 — de modo a calcular a nova posição do míssil na tela.

As linhas 2020 a 2040 averiguam o ponto em que o míssil está na tela e também a blindagem necessária para bloquear o míssil — **MA** é o ângulo de deslocamento do míssil.

A linha 2050 inicia a trajetória do míssil a partir da posição da nave inimiga. E a linha 2060 coloca o foguete na tela antes de a linha 2070 definir a variável sinalizadora **AF = 1,** que diz ao programa se um míssil foi disparado. Uma vez concluídos esses passos, a subrotina termina.

A sub-rotina final vai da linha 3000 à 3070. Digite as linhas, mas novamente nada acontecerá se você rodá-las.

```
3000 PUT(MX,MY)-(MX+4,MY+4),BL,
PSET
3010 LET MX=MX+DX:LET MY=MY+DY
3020 IF MX>110 AND MX<140 AND M
Y>79 AND MY<108 THEN GOTO 3050
3030 PUT(MX,MY)-(MX+4,MY+4),BO,
OR
3040 RETURN
3050 IF SH(MA)=0 THEN GOTO 3070
3060 LET AF=0:RETURN
3070 CLS:PRINT @256,"BANG...SUA
 BLINDAGEM ESTAVA DESLIGADA!"
```

O míssil é apagado pela linha 3000, e sua nova posição é computada pela linha 3010. A linha 3020 verifica se o míssil já atingiu a blindagem. Se isso não aconteceu, o programa pula rapidamente para a linha 3050, verificando se a blindagem correta estava posicionada. Quando nenhuma blindagem é encontrada no caminho do foguete, o programa limpa a tela e termina com a mensagem: "BANG! ... SUA BLINDAGEM ESTAVA DESLIGADA".

Se o míssil ainda não atingiu o ponto esperado para alcançar as blindagens da estação espacial, a linha 3050 o colocará rapidamente em nova posição com o comando **PUT.**

A linha 3040 finaliza a rotina. Para terminar, as seguintes linhas completam o nosso programa:

```
150 SCREEN 1,1
160 IF AF=0 THEN GOSUB 2000 ELS
E GOSUB 3000
170 GOSUB 1000
180 FOR J=1 TO 4
190 LET PE=PEEK(338+J):IF PW<25
 THEN LET PE=225
200 IF (255-PE)/16<>SH(J) THEN
LET SH(J)=1-SH(J):CIRCLE(127,95
),16,5*SH(J),1,(J+2)/4,(J+3)/4
210 IF SH(J)=1 THEN LET PW=PW-2
```

O programa para o Spectrum utiliza diversos recursos novos que não foram explicados em capítulos anteriores. Por isso, é aconselhável estudar um pouco esses recursos e fazer algumas experiências.

Como de costume, você pode ir checando se os trechos que acabou de entrar funcionam corretamente. O primeiro grupo de linhas define o gráfico da nave extraterrestre e o desenha na tela, quando executado.

```
10 BORDER 0: PAPER 0: INK 6:
BRIGHT 1: CLS
20 FOR n=USR "a" TO USR "b"+7
: READ a: POKE n,a: NEXT n
200 LET ax=INT (RND*32)
210 LET ay=INT (RND*21)+1
220 IF ax>11 AND ax<21 AND ay>
6 AND ay<16 THEN  GOTO 200
490 PRINT  INK 4,AT ay,ax;CHR$
144
800 DATA 60,126,219,219,126,60
,90,153,0,0,24,60,60,24,0,0
```

As linhas 20 e 800 definem a nave e seu míssil (que ainda não é visível). Elas utilizam a técnica de colocar em uma área de memória RAM uma série de blocos gráficos elementares, definidos dentro das declarações **DATA.** Essa área é chamada de *área de gráficos definíveis pelo usuário* (ou UDG, abreviatura em inglês da expressão *user = defined-graphics).* Os comandos **USR "a"** e **USR "b"** definem, para o laço **FOR...NEXT,** os endereços inicial e final dessa área. Os comandos **POKE** fazem esse trabalho.

As linhas 200 e 210 colocam a nave espacial em uma posição aleatória na tela, e a linha 490 a imprime. (O **PRINT CHR$ 144** nessa linha mostra o caractere gráfico associado à tecla A).

A linha 220 pode parecer estranha nesse estágio mas, à medida que o programa progredir, você verá que esta é uma maneira de impedir que a nave inimiga apareça inesperadamente no meio de sua estação espacial, trazendo pânico e destruição.

No momento, você pode sentir vontade de omitir a linha 10, pois pode ser mais difícil ler o programa listado em amarelo sobre uma tela negra. Se quiser fazer isso, lembre-se de reintegrá-la mais tarde, ou a linha 640 (explicada mais adiante) não funcionará.

CONSTRUA A ESTAÇÃO ESPACIAL

Estas poucas linhas desenharão a estação espacial:

```
110 PRINT AT 10,15;"4 1";AT 12
,15;"3 2"
120 PLOT 132,107: DRAW 25,-25:
DRAW -25,-25: DRAW -25,25:
DRAW  25,25
130 PLOT 107,82: DRAW 50,0:
PLOT 132,57: DRAW 0,50
```

Por enquanto, a estação espacial ainda parece bastante primitiva. Se você quiser projetar e entrar uma mais adequada ou mais bonita, precisará apenas de duas adições ao programa:

- Uma linha extra parecida com a linha 20, mas que comece com **USR "c"** e que continue por tantas letras do alfabeto quanto o tamanho de sua estação espacial necessitar.
- Um conjunto de declarações **DATA** em uma ou mais linhas extras no final do programa, listando os códigos gráficos que compõem a nova estação espacial na tela do computador.

MOVIMENTE O MÍSSIL

A próxima tarefa é imprimir na tela o míssil da nave E.T. e traçar sua trajetória em direção à estação espacial.

```
150 LET mf=0
300 IF mf=1 THEN  GOTO 400
310 IF RND<.9 THEN  GOTO 420
320 LET mf=1: LET my=ay: LET m
x=ax: LET fy=11-my: LET fx=16-
mx
330 LET b=1: IF ABS fy>ABS fx
THEN  LET b=2
340 IF b=1 THEN  LET sx=SGN fx
: LET sy=SGN fy*ABS (fy/fx)
350 IF b=2 THEN  LET sy=SGN fy
: LET sx=SGN fx*ABS (fx/fy)
400 PRINT AT my,mx;" ": LET my
=my+sy: LET mx=mx+sx: PRINT
INK 5;AT my,mx;CHR$ 145: IF my
>10 AND my<12 AND mx>15 AND mx
<17 THEN  GOTO 700
620 IF RND>.9 THEN  PRINT AT a
y,ax;" ": GOTO 200
630 IF mf=0 THEN  GOTO 300
650 GOTO 300
700 CLS : PRINT  FLASH 1;
PAPER 2;AT 10,1;"BANG!  Voce e
sta sem  protecao!"
```

Toda esta seção do programa, como você pode ver a partir da linha 650, é um laço que o computador percorre diversas vezes sempre que a nave inimiga aparece.

A linha 150 descreve todo o cenário: não existe nenhum míssil vindo em sua direção (até agora...).

A linha 310 decide se a nave lançará um míssil em sua direção, dentro dessa repetição do laço (existe uma chance em nove de que irá disparar).

Se houver um míssil, a linha 320 o colocará na posição inicial (my, mx), no lugar mais óbvio — ou seja, no lugar em que a nave E.T. se encontra no momento (ax, ay). A parte do meio da linha 400 imprime o míssil, usando o caractere gráfico CHR$ 145 (correspondente à tecla B).

O trecho do programa a partir da metade da linha 320 até a linha 400 é um pouco "manhoso". O que ele faz é tomar as coordenadas do ponto médio da estação espacial e as coordenadas referentes à posição atual da nave e subtrair estas das primeiras, de modo a disparar o míssil na direção exata da estação espacial (numa reta passando por dois pontos).

Como alguns dos números envolvidos nesses cálculos podem ser negativos (em movimentos para a esquerda e para baixo) e outros positivos (em movi-

mentos para a direita e para cima), você vai achar difícil entender como o programa funciona, caso não compreenda o que são ABS e SGN, que foram definidos no capítulo anterior. Mas eis aqui algumas dicas para esclarecer o assunto:

A segunda metade da linha 320 deduz a posição atual do míssil (my, mx) do ponto central da estação espacial (posição na tela 11, 16) e chama as coordenadas resultantes (fy e fx).

As linhas 330 e 350, utilizando ABS e SGN, acrescentam um fator de correção de trajetória (sy, sx) para fy e fx. A linha 400 começa imprimindo um espaço em branco sobre a posição anterior do míssil. Em seguida, soma-se sx e sy às coordenadas da posição anterior, de modo a preparar a parte restante da linha 400 para imprimir o míssil em sua nova posição, apenas um passo distante da anterior.

A NAVE INIMIGA

Agora que a nave inimiga já disparou seu míssil, é hora de fazê-la mover-se; acrescente essas linhas:

```
420 LET xx=ax: LET yy=ay: LET
m=INT (RND*4)
430 IF m=0 AND ax<31 THEN  LET
xx=ax+1
440 IF m=1 AND ax>0 THEN  LET
xx=ax-1
450 IF m=2 AND ay<21 THEN  LET
yy=ay+1
460 IF m=3 AND ay>1 THEN  LET
yy=ay-1
470 IF xx>11 AND xx<21 AND yy>
6 AND yy<16 THEN  GOTO 490
480 PRINT AT ay,ax;" ": LET ax
=xx: LET ay=yy
```

Em primeiro lugar o Spectrum decidirá em que direção a nave E.T. se movimentará. Uma vez que isso seja feito, por um número aleatório na linha 420, os objetivos das linhas 430 a 460 tornam-se óbvios. Estas são as linhas convencionais de animação gráfica. A linha 470 mantém a nave fora da estação. A linha 400 (inserida anteriormente) registra um impacto, direcionando o programa para a linha 700, caso o míssil atinja o meio da estação. Se você não sabe por que essa linha usa **IF my>10 AND my<12,** ao invés de um 11 (mais simples) e **IF mx> 15 AND mx<17,** em vez de 16, basta lembrar-se do seguinte: embora o computador imprima a nave extraterrestre em uma posição com coordenadas inteiras, os números calculados a cada passo do movimento são uma série de *frações decimais.* Assim, torna-se remota a chance de eles tornarem-se 11 e 16.

Finalmente, depois de mais ou menos dez repetições do laço, a linha 620 apaga a nave inimiga de sua posição e começa tudo de novo na linha 200.

CONSTRUA AS BLINDAGENS

As linhas seguintes constroem as blindagens destinadas a bloquear o avanço do míssil.

```
140 PLOT  INVERSE 1;132,122
500 DIM a(4)
510 LET a$=INKEY$: IF a$=""
THEN  GOTO 600
520 IF a$="1" THEN  LET a(1)=1
530 IF a$="2" THEN  LET a(2)=1
540 IF a$="3" THEN  LET a(3)=1
550 IF a$="4" THEN  LET a(4)=1
560 LET fu=fu-1
600 DRAW  INK a(1)*4, INVERSE
1-a(1),40,-40: DRAW  INK a(2)*
4, INVERSE 1-a(2),-40,-40:
DRAW  INK a(3)*4, INVERSE 1-a(
3),-40,40: DRAW  INK a(4)*4,
INVERSE 1-a(4),40,40
640 IF ATTR (my,mx)=68 THEN
PRINT AT my,mx;" ": LET mf=0
```

À primeira vista também existe alguma coisa estranha com essas linhas. Quatro blindagens, mas apenas uma posição *PLOT* para traçar as linhas?

De fato, o programa utiliza a cor do traço **(INK),** que é a mesma do fundo **(PAPER),** para traçar um losango. Somente quando você pressionar uma das teclas numeradas de 1 a 4, uma parte do losango mudará de cor e aparecerá na tela. A linha 640 utiliza o ATTR 68 — o número da cor das blindagens — para repelir o míssil, se ele acertar a blindagem, apagando-o da tela.

LIMPEZA FINAL

As linhas restantes são muito fáceis de se compreender. Elas estabelecem o suprimento do combustível na linha 100 e o fazem desaparecer até que, no meio da linha 510 (já modificada), as blindagens tornam-se inativas. Lembre-se de reintegrar a linha 10.

```
100 PRINT  PAPER 2; INK 6;AT 0
,0;" COMBUSTIVEL "
160 LET fu=100
510 LET a$=INKEY$: IF a$=""
 THEN  GOTO 600
560 LET fu=fu-1
610 PRINT  PAPER 3; INK 7;AT 0
,13;" ";fu;" "
```

MSX

```
10 R=RND(-TIME)
15 SCREEN 2,0
30 DEF FN Z(X)=SGN(X)*SQR(V*V*X
*X/((127-AX)*(127-AX)+(95-AY)*(
95-AY)))
32 LET PW=250
50 FOR I=0 TO 7
55 READ A:LET A$=A$+CHR$(A)
59 NEXT I
60 SPRITE$(1)=A$
61 PUT SPRITE 1,(8,0)
65 GOTO 65
250 DATA 24,126,90,126,126,195,
129,129
```

Digitando e executando esse programa, você verá a nave espacial extraterrestre aparecer na tela.

A linha 10 serve apenas para introduzir o gerador de números aleatórios, usando o conteúdo do relógio (TIME).

A linha 30 utiliza uma palavra chave que você ainda não viu: **DEFFN,** que *significa DEFine a FuNction,* ou seja, *definição de função.* Se você precisar utilizar com freqüência uma longa expressão matemática no programa, a função predefinida o poupará de digitá-la diversas vezes, em diferentes pontos do programa. Ela funciona, portanto, como uma espécie de sub-rotina em uma linha só. A expressão matemática na linha 30 é chamada de **FNZ** e será utilizada mais tarde no programa para movimentar um míssil em sentido diagonal na tela. A linha 32 estabelece na tela a posição final de um indicador de combustível (variável **PW**).

As linhas 50 a 61 desenham a nave E.T. na tela, lendo os códigos gráficos correspondentes através de instruções

READ e **DATA** na linha 250, e colocando-os na página de vídeo, no canto superior esquerdo da tela, com o comando **PUT SPRITE**. A linha 60 usa a função SPRITE para definir este padrão gráfico e armazená-lo no "sprite" número I (por enquanto, não explicaremos detalhadamente o que é

um sprite: veja MICRODICAS na página 108). A linha 61 é temporária, pois apenas mostra o resultado do SPRITE, sendo retirada depois.

A linha 65 também é temporária. Tudo o que ela faz é manter a tela "congelada", de modo que se possa ver o resultado do processamento feito pelas linhas anteriores. Ela será removida mais tarde, quando o resto do jogo for acrescentado ao programa.

DESENHE OS MÍSSEIS

Agora remova as linhas 61 e 65, digitando os números 65, seguido de <ENTER>, e 61. Também seguido de <ENTER>. Acrescente depois essas linhas e rode o programa.

```
70 FOR J=0 TO 7
75 READ B:LET B$=B$+CHR$(B)
79 NEXT J
80 SPRITE$(0)=B$
81 PUT SPRITE 0,(16,13)
85 GOTO 85
260 DATA 0,0,0,8,28,62,28,8
```

O míssil que será lançado pelo E.T. é lido em um conjunto de códigos em **DATA** e definido como o sprite número 0, nas linhas 70 a 80. O comando **PUT** coloca-o no canto superior esquerdo da tela para você ver, na linha 81. Assim, as linhas 81 e 85 são temporárias.

A ESTAÇÃO ESPACIAL

Remova as linhas 81 e 85 como anteriormente, e acrescente essas linhas, executando em seguida o programa:

```
90 LET AX=INT(RND(1)*200)-1:LET
 AY=INT(RND(1)*100)+5
110 CIRCLE (127,95),12,15:DRAW
"BM127,95;C15S48NUNLNDNR"
115 GOTO 115
```

A linha 90 escolhe uma posição aleatória para a nave E.T. A linha 110 desenha a estação espacial, que é simplesmente um círculo (comando **CIRCLE**). O comando **DRAW**, no final dessa linha, traça uma cruz bem no meio da estação espacial. Como foi dito anteriormente, o comando **DRAW** pode ser imaginado como uma sucessão de declarações **LINE.** Uma explicação mais detalhada será encontrada em outra lição, mais adiante.

A linha 115 é temporária.

O INDICADOR DE COMBUSTÍVEL

Remova a linha 115 do mesmo modo que removeu as linhas 61, 65, 81 e 85. Acrescente essas linhas e aparecerão na tela maiores detalhes gráficos.

```
120 DRAW "BM 131,87;S4D5BD6BR3B
D4U5G4R3BDBL12R3U2NL3U2NL3BU10B
L3R3D2L3D2R3"
130 DRAW "BM5,1;L4D2NR4D2BE4BR2
D4R3U4BR5L3D2NR3D2R3BE4D4R3"
140 LINE(30,1)-(PW,5),10,BF
145 GOTO 145
```

A linha 120 traça um número na estação espacial, que corresponde ao número de blindagens. Na linha 130 mostramos a palavra FUEL (combustível). Infelizmente o MSX não consegue escrever caracteres normais numa tela gráfica de alta resolução; por isso letras e números devem ser literalmente "desenhados" com comandos **DRAW.**

A linha 140 exibe o indicador cheio de combustível para as blindagens. Essa linha emprega um método rápido para traçar retângulos. Usamos a instrução **LINE** para traçar uma linha no alto da tela, indo da posição extrema esquerda (coordenadas 30,1) até a posição atual do combustível (assinalada pela variável **PW**). O número 10 diz ao MSX para traçar essa linha em cor branca. BF é uma abreviação para **box fill** e preenche o retângulo com a cor utilizada pela linha original. Se você quer traçar um retângulo vazio, utilize B ao invés de BF.

Estão agora completos os gráficos de alta resolução para o jogo. O resto do programa está relacionado com o movimento do E.T. e do míssil e a ativação das blindagens.

A NAVE INIMIGA

Existem três sub-rotinas a serem acrescentadas ao programa. A primeira diz respeito ao movimento da nave inimiga. Digite-a, mas não a rode ainda porque nada acontecerá.

```
1000 LET LX=AX:LET LY=AY
1010 IF INT(RND(1)*10)=1 THEN L
ET AX=RND(1)*248-1:LET AY=INT(R
ND(1)*178)+5
1020 LET AX=AX+INT(RND(1)*15)-8
:LET AY=AY+INT(RND(1)*15)-8
1030 IF AX>103 AND AX<144 AND A
Y>71 AND AY<112 THEN LET AX=LX:
LET AY=LY
1040 IF AX<0 THEN LET AX=-AX
1050 IF AX>248 THEN LET AX=497-
AX
1060 IF AY<6 THEN LET AY=12-AY
1070 IF AY>184 THEN LET AY=369-
AY
1080 PUT SPRITE 1,(AX,AY)
1100 RETURN
```

A nave E.T. é controlada de forma semelhante à base de mísseis na lição das páginas 28 a 33. *(Apontar...Fogo!)*. A linha 1000 iguala as coordenadas da última posição da nave às coordenadas da posição atual, antes que ela seja movimentada.

A linha 1010 escolhe um número ao acaso, provocando uma brusca mudança de posição da nave inimiga na tela ("hiperespaço"). Se esse número aleatório for I (ou seja, em 10% das vezes, em média), o E.T. saltará para uma nova posição na tela. Se for diferente de I, será escolhida uma nova posição para a nave, distante entre -7 e +7 pontos na direção X (da esquerda para a direita), e entre -7 e +7 pontos na tela, na direção Y (acima e abaixo) — veja a linha 1020. A linha 1030 impede que a nave E.T. superponha-se ao desenho da estação espacial, enquanto as linhas 1040 a 1070 impedem que ela saia para fora da tela.

A nave invasora é colocada na nova posição pela linha 1080. A linha 1100 faz retornar a sub-rotina.

DISPARE OS MÍSSEIS

A próxima sub-rotina serve para decidir, em primeiro lugar, se dispara ou não o míssil. Sendo positivo (isto é, se o disparo ocorrer), então ela estabelecerá a posição do míssil na tela e determinará a blindagem que irá bloqueá-lo.

```
2000 IF INT(RND(1)*7)<>1 THEN R
ETURN
2010 LET V=INT(RND(1)*8)+5
2015 LET DX=FNZ(127-AX):LET DY=
FNZ(95-AY)
2020 IF DX<=0 AND DY>=0 THEN LE
T MA=1:GOTO 2050
2030 IF DX<=0 AND DY<=0 THEN LE
T MA=4:GOTO 2050
2040 IF DX>=0 AND DY<=0 THEN LE
T MA=3 ELSE MA=2
2050 LET MX=AX:LET MY=AY
2060 PUT SPRITE 0,(MX,MY)
2070 LET AF=1:RETURN
```

A linha 2000 decide se dispara um míssil. Existem seis chances em uma de que ele seja disparado; mas, se já há um míssil na tela, o programa não pode lançar outro. Se um foguete não puder ser disparado, a sub-rotina terminará.

A linha 2010 estabelece a extensão de deslocamento do míssil na tela — você pode pensar em **V** como se fosse a sua velocidade. O conteúdo de **V** é passado para a função **FNZ** — definida na linha 30 — de modo a calcular a nova posição do míssil na tela.

As linhas 2020 a 2040 investigam o ponto em que o míssil está na tela e também que blindagem será necessária para bloqueá-lo — MA é o ângulo de deslocamento do míssil.

A linha 2050 inicia a trajetória do míssil a partir da posição onde está a nave inimiga. A linha 2060 coloca o míssil na tela antes de a linha 2070 definir a variável sinalizadora **AF = I**, que diz ao programa que um míssil foi disparado. A sub-rotina termina.

A sub-rotina final vai da linha 3010 à 3080. Digite as linhas, mas observe que novamente não acontecerá nada se você rodá-las nesse estado.

```
3010 LET MX=MX+2*DX:LET MY=MY+2
*DY
3020 IF MX>110 AND MX<140 AND M
Y>79 AND MY<108 THEN GOTO 3050
3030 PUT SPRITE 0,(MX,MY)
3040 RETURN
3050 IF SH(MA)=0 THEN GOTO 3070
3055 PUT SPRITE 0,(-1,-1)
3060 LET AF=0:RETURN
3070 SCREEN 0
3080 LOCATE 0,11:PRINT "BANG!!
- Sua blindagem estava desligad
a"
```

A nova posição do míssil é computada pela linha 3010. A linha 3020 checa se o míssil já atingiu a blindagem. Se isso não acontecer, o programa pula para a linha 3050, com o objetivo de verificar se a blindagem correta estava posicionada. Se não for encontrada nenhuma blindagem no caminho do foguete, o programa limpará a tela e terminará com a mensagem: "BANG! ... SUA BLINDAGEM ESTAVA DESLIGADA".

Se o míssil ainda não atingiu o ponto esperado para as blindagens da estação espacial, então a linha 3050 o coloca em nova posição com o comando **PUT**.

A linha 3040 finaliza a rotina. Finalmente, as seguintes linhas completam o nosso programa:

```
160 IF AF=0 THEN GOSUB 2000 ELS
E GOSUB 3010
170 GOSUB 1000
180 FOR J=1 TO 4
190 K$=INKEY$:IF K$="" THEN K$=
"*"
195 LET PE=ASC(K$)-48
200 IF PE=J THEN SH(J)=1 ELSE S
H(J)=0
201 IF SH(J)<>1 THEN LET SH(J)=
0
205 CIRCLE(127,95),18,15*SH(J),
6.2*(J-1)/4,6.2*(J)/4
210 IFSH(J)=1 THEN LET PW=PW-2
220 NEXT
230 LINE (PW,1)-(PW,5),0,BF
240 GOTO 160
```

Antes de rodar o programa, remova as linhas 115 e 145. Agora que a linha 145 foi removida, você não verá o que está se desenvolvendo na tela durante a definição dos blocos gráficos. Após o programa ter sido rodado, ocorrerá uma curta pausa, antes que uma exposição completa apareça na tela.

A linha 160 verifica se um míssil foi lançado. A equação **AF = 0** indica que nenhum míssil foi disparado, e o programa pula para a sub-rotina que se refere ao lançamento do míssil — que começa na linha 2000 — ou pula para a sub-rotina que movimenta o míssil; esta começa na linha 3010. Depois, a nave E.T. é movimentada. A linha 170 faz com que o programa salte para a sub-rotina que tem início na linha 1000.

A parte do programa das linhas 180 a 220 diz respeito à ativação das blindagens. A linha 190 checa qual tecla está sendo pressionada. Se a tecla for um número de 1 a 4, a linha 2000 desenhará a blindagem; se alguma das blindagens for acionada, a linha 210 diminuirá o combustível.

A linha 230 traça um retângulo negro no final do indicador do combustível, dando a impressão de que o suprimento de combustível está acabando, à medida que o valor em **PW** diminui.

Finalmente, a linha 240 repete tudo novamente, indo para a linha 160.

O QUE É UM *SPRITE*?

Um *sprite* é um tipo de caractere de alta resolução gráfica definido pelo usuário. Ele é utilizado para acrescentar movimento aos gráficos de alta resolução do MSX.

O uso do sprite fornece um aspecto contínuo ao movimento de figuras na tela. Esse movimento pode ser facilmente programado ponto por ponto, como se o objeto que se move fosse apenas um caractere. Além disso, um sprite pode ter até 16x16 pontos de alta resolução, enquanto um caractere comum tem 8x8 pontos. O contorno ou a forma do caractere ou do sprite é definido por pontos "acesos" que assumem a cor de frente, e por pontos "apagados" que assumem a cor de fundo.

É possível definir até 256 sprites pequenos (8x8 pontos) e até 64 sprites grandes (16x16), com padrões diferentes. Todavia, só podem ser mostrados quatro sprites diferentes na mesma linha horizontal de alta resolução. Sprites podem ser agrupados para formar figuras maiores; podem ser sobrepostos para obtermos efeitos tridimensionais; podem, ainda, ter seu tamanho dobrado. Por outro lado, é possível saber quando dois sprites colidiram; tudo isso faz deles caracteres de alta resolução extremamente versáteis.

Programar sprites é realmente mais fácil que utilizar caracteres definidos pelo usuário (compatíveis com o Sinclair Spectrum) ou funções como **PUT**, **GET** (compatíveis com o TRS-Color) e **DRAW** (compatíveis com o Apple II), principalmente quando se deseja uma animação mais sofisticada. Não é necessário, por exemplo, seguir o rastro de um sprite, apagando as posições já ocupadas por ele. Os sprites são compatíveis com outros modos de utilização da tela do MSX: texto em 32 colunas (SCREEN 1) e modo multicor (SCREEN 3). Não se pode, contudo, utilizá-los em modo texto de 40 colunas (SCREEN 0).

NO CORAÇÃO DE UM MICRO

- O QUE É E O QUE FAZ UMA UCP
- COMUNIQUE-SE COM O MICROPROCESSADOR
- OS REGISTROS INTERNOS
- OS SINALIZADORES
- LOCALIZAÇÕES DE MEMORIA

Para entender a linguagem de máquina é necessário aprender primeiro como funciona o microprocessador, que é, ao mesmo tempo, o coração e o cérebro de um computador pessoal.

Como você já sabe, os computadores possuem memória. Mas eles não se limitam a lembrar-se das coisas; na verdade, essas máquinas também são capazes de manipular e processar aquilo que memorizam, através da unidade central de processamento (UCP), que é formada por um único circuito integrado *(chip)*, conhecido como *microprocessador*.

Esse circuito integrado é muito mais complexo do que um circuito de memória. As memórias são dispositivos relativamente passivos, que armazenam padrões binários e os desenvolvem. O microprocessador, entretanto, realiza funções bem mais "inteligentes". Ele pode adicionar, subtrair, movimentar informação de uma locação de memória para outra, e deslocar e alternar seus padrões de bits. A programação em linguagem de máquina tem por objetivo "dizer" ao microprocessador (UCP) como ele deve proceder para processar dados e informações, executando uma grande variedade de *operações*.

O CÉREBRO DA MÁQUINA

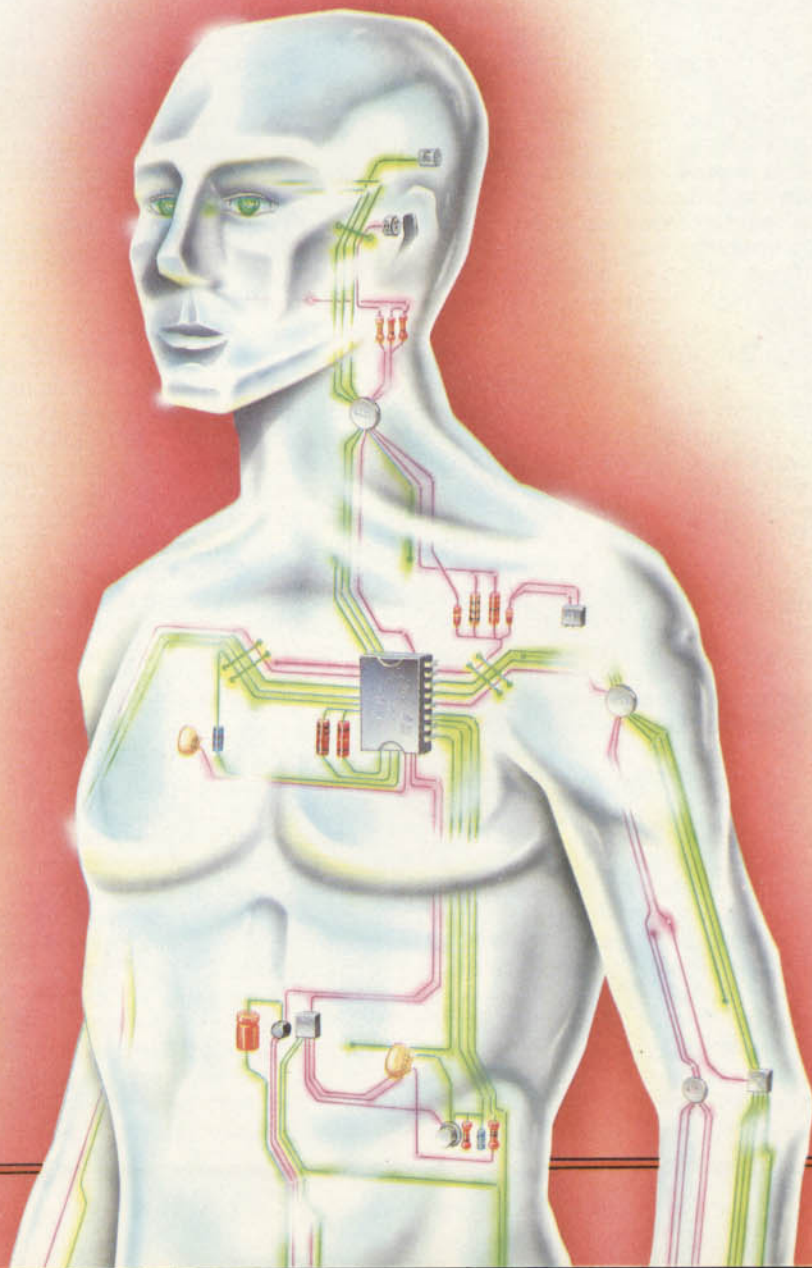
Tudo que acontece no computador está literalmente sob controle direto do microprocessador. Verdadeiro "cérebro", ou centro de controle da máquina, ele efetua todas as operações aritméticas, copia dados de uma locação de memória para outra, atribui funções para áreas de memória e executa os programas. É com o microprocessador que você está se comunicando quando escreve seus programas em código de máquina. Dentro dele existem inúmeras locações especializadas de memória, chamadas *registros*. Cada registro tem uma função específica. E a seleção dos que serão utilizados depende inteiramente das instruções que você fornece quando escreve o seu código de máquina.

Uma das coisas mais importantes sobre os registros é que eles não possuem endereços. Isso faz com que eles se tornem de utilização muito mais rápida.

OS REGISTROS INTERNOS

As diferentes linhas de computadores pessoais utilizam, geralmente, microprocessadores de marcas diferentes. Assim, os micros das linhas Sinclair, TRS-80 e MSX, por exemplo, empregam o microprocessador de oito bits de comprimento de palavra chamado Zilog Z-80A (este é o seu nome comercial). Já os micros da linha Apple II utilizam um microprocessador também de oito bits mas inteiramente diferente, o 6502.

A complicação para o futuro programador de linguagem de máquina começa aqui: cada um dos diversos chips de microprocessadores citados possui um conjunto diferente de registros internos e de instruções em código de máquina. Em geral, o que se aprende para um ti-

po de chip não se aplica para os outros.

Felizmente, alguns dos registros têm funções similares em todas as máquinas. Os microprocessadores de oito bits possuem um registro chamado PC (Program Counter) de dezesseis bits: é o *contador de programa.* Este armazena o endereço do byte seguinte do código de máquina de um programa a ser executado. Ele "avisa" ao computador onde se encontra, dentro de um programa. E, assim que o byte do programa de código de máquina é executado, o registro PC é aumentado de um.

Todos os microcomputadores de oito bits também têm, pelo menos, um outro registro, chamado *acumulador* de oito bits ou registro A. É o registro utilizado pela UCP para efetuar operações aritméticas. Se você quiser, por exemplo, acrescentar um número a outro (ou subtraí-lo dele), uma das duas parcelas deverá estar no acumulador. Isso também afeta a multiplicação e a divisão — que são apenas combinações de operações lógicas como as adições e as subtrações.

Cada uma dessas máquinas possui também um par de *registros de indexação.* Nos microprocessadores Z-80A estes são registros de dezesseis bits que contêm endereços. Tais endereços podem ser aumentados, diminuídos ou modificados aritmeticamente no acumulador, de modo a fornecer outros endereços. Os micros da linha Apple (microprocessador 6502) possuem registros de indexação de oito bits cujos conteúdos são incorporados a um endereço para fornecerem um endereço adicional.

Conhecido como *endereçamento indexado,* esse tipo de operação é mais comumente utilizado para ler dados de uma tabela, byte por byte. O programa começa com o endereço do primeiro byte da tabela e, a seguir, percorre seu caminho aumentando o registro de indexação de um em um.

LEVANTE A BANDEIRA

O sinalizador (registro-bandeira, ou *flag-register*, em inglês, numa analogia à bandeira dos taxímetros) é outro registro existente em todos os microprocessadores de 8 bits. Um registro sinalizador tem normalmente oito bits, que podem estar "ligados" (bit 1), ou "desligados" (bit 0), dependendo do resultado de certas operações. Cada bit de um registro sinalizador é chamado de *bandeira,* ou sinalizador.

Todo sinalizador tem um significado próprio. Por exemplo, o bit número 0, ou seja, o bit menos significativo do registro sinalizador (o último bit à direita na convenção normal de escrita), é o bit de transporte ou sinalizador C (*carry,* em inglês). Se a UCP efetuar uma adição, uma subtração, ou qualquer outra operação lógica ou aritmética que exija um bit para ser emprestado ou transportado (o "vai-um" das operações aritméticas), então o sinalizador C será estabelecido em 1.

Os outros bits indicam se o resultado de uma operação foi zero *(o sinalizador de zero)* ou negativo *(o sinalizador de sinal),* ou ainda se um registro foi excedido em capacidade porque o número era muito grande *(o sinalizador de excesso,* ou *overflow).* Esses sinalizadores são comuns a todas as máquinas, embora não estejam necessariamente na mesma posição no registro sinalizador. Existem também sinalizadores específicos para cada máquina.

Os sinalizadores são utilizados em instruções de desvio condicional em linguagem de máquina — o equivalente em BASIC ao **IF...THEN.** No programa em código de máquina, podemos escrever instruções que farão com que a máquina salte ou desvie se um sinalizador estiver em 0 ou em 1, por exemplo.

A PILHA

Comum a todos os computadores, o *indicador da pilha* (em inglês: *stack-pointer)* é um registro destinado a reter o endereço de uma locação de memória RAM. Esse endereço marca o fim de uma área de memória, como todos os indicadores fazem; mas, aqui, essa é uma área especializada de memória, de uso exclusivo da UCP e conhecida como *pilha.* Ela é uma área de rápido acesso, onde a informação pode ser temporariamente armazenada e chamada de volta sem precisar dos procedimentos normais de endereçamento.

A pilha da UCP funciona de modo parecido a uma pilha comum de papéis: toda informação nova que se quer acrescentar a ela deve ser colocada em seu topo. Inversamente, quando se quer tirar dela alguma coisa, é necessário começar pelo topo, retirando as folhas mais altas (não vale tirar um papel do meio ou da parte mais baixa da pilha). A regra é: o último a entrar é o primeiro a sair (em inglês: *last in, first out*). Por isso, esse tipo de pilha é chamado de LIFO. Existem outras espécies como a pilha FIFO *(first in, first out).* A fila de um supermercado é um exemplo de pilha FIFO...

Uma das funções do microprocessador consiste em colocar dados, endereços ou códigos no topo da pilha da memória. Se ele for instruído para retirar algo da pilha, começará sempre pelos itens mais altos. Isso significa que você não precisa especificar a locação de memória que lhe interessa, ou sua posição na pilha. Existe apenas uma locação do topo de cada vez; assim, o microprocessador não se confundirá.

Mas você, o programador, pode ficar extremamente confuso, se não prestar atenção! É importante lembrar-se de tudo o que existe em cada locação de memória da pilha, quando se estiver fazendo um programa que a utiliza, e certificar-se de que o dado desejado está no topo da pilha.

Agora que você já domina o conceito de pilha LIFO, podemos revelar-lhe um segredo: em todas as máquinas citadas aqui, a base da pilha é dirigida para o topo da RAM, com o resto da pilha acumulado abaixo, item por item. Assim, na verdade, você introduz e tira coisas da parte inferior da pilha!

Isto quer dizer que, quando um novo item é introduzido, o indicador de pilha é diminuído, e, quando um item é tirado, o indicador é aumentado.

Apesar disso, o conceito de pilha se mantém verdadeiro, com a base da pilha fixada por uma variável de sistema, e os itens dos dados adicionados ou subtraídos a partir de um final livre, um de cada vez.

SUB-ROTINAS

Uma das funções mais comuns de uma pilha é manter a UCP a par de seu lugar em um programa quando uma sub-rotina for executada. Isso pode parecer simples, mas devemos nos lembrar de que uma sub-rotina pode chamar outra; esta, uma outra, e assim por diante. Quando o computador atinge uma instrução que lhe diz para saltar para uma sub-rotina, o conteúdo do contador de programa nesse momento (isto é, onde o programa se encontra no instante em que a instrução de desvio ocorre) é copiado no topo da pilha.

O endereço inicial da sub-rotina é então escrito no registro contador de progama. À medida que cada instrução é executada em uma sub-rotina, o contador de programa é incrementado da maneira normal, até que atinja uma instrução de retorno. Então, o endereço que está no topo da pilha é retirado e colocado de volta no registro contador de programa. Esse processo permite que a UCP reinicie a execução do programa exatamente no ponto onde estava antes de chamar a sub-rotina e passe a incrementar o contador de programa nova-

mente. Isso explica por que, em programas BASIC, cada sub-rotina encaixada dentro de uma outra deve estar contida completamente naquela que a precede; do contrário, os endereços incorretos de partida seriam colocados no topo da pilha. O mesmo acontece com os laços de repetição, tipo **FOR...NEXT,** cujos endereços de retorno no programa são armazenados na pilha. Agora que você domina os conceitos básicos da programação de códigos de máquina, descubra como eles se relacionam ao microprocessador específico de seu computador.

O microprocessador dessas máquinas é o Zilog Z80A, de oito bits. Ele tem características exclusivas, como o fato de contar com um grande número de registros internos (21, para ser exato). Estes são também chamados de registros do usuário, porque podem ser controlados diretamente pelo usuário ou pelo programador da máquina.

Existem dois acumuladores de oito bits: A e A'. Esses acumuladores não podem ser utilizados ao mesmo tempo, mas você pode comutá-los para desempenharem simultaneamente duas operações. Contudo, não é aconselhável tentar isso no ZX81. Se a máquina estiver na modalidade *SLOW* (lenta) você poderá perder o conteúdo da tela.

O registro sinalizador F também possui oito bits. O bit na posição 0 é o sinalizador de transporte C; o bit na posição 1 é o sinalizador de subtração N, o qual entra em ação apenas quando operações codificadas em BCD (*decimal codificado em binário* — uma representação especial de números decimais em binário) são utilizadas; o bit 2 é o sina-

lizador de paridade/excedente P/V; o bit 4 é o sinalizador de meio transporte H, que também é utilizado em BCD; o bit 6 é o sinalizador de 0 Z, e o bit 7 é o sinalizador de sinal S. Existe um registro sinalizador alternativo F' que é comutado junto com A'.

Existem seis registros de propósitos gerais — B, C, D, E, H e L — que tanto podem ser utilizados separadamente, como registros de oito bits, quanto em pares, como registros de dezesseis bits (neste caso, são chamados coletivamente de registros BC, DE e HL). Também existe um conjunto alternativo: B 'C', D 'E' e H 'L'. O HL e os registros alternativos H 'L' podem igualmente ser utilizados como acumuladores de dezesseis bits. Contudo, os registros H 'L' devem ter seus valores anteriores restaurados antes de retornarem para BASIC.

O indicador de pilha do Z-80 tem dezesseis bits, assim como os registros de indexação IX e IY. Estes últimos retêm endereços que podem ser aumentados ou deslocados por meio da adição ou subtração de números conhecidos como "deslocamentos" *(off-sets)*, destinados a fornecer um outro endereço. O IY aponta para o centro das variáveis de sistemas e é utilizado pelas rotinas ROM para indexá-las. Se o registro IY for acionado, seu valor original deverá ser restaurado mais tarde ou o computador não funcionará.

Existem mais dois registros de propósitos especiais no microprocessador Z-80: o I e o R. O registro I armazena parte do endereço utilizado para iniciar "rotinas interrompidas". São rotinas que interrompem o curso normal do seu programa de código de máquina a cada 1/50 de segundo, para desempenhar alguma função vital tal como a varredura do teclado.

O registro R renova a memória dinâmica *(refresh)*, mas você não poderá utilizá-lo no Spectrum ou no ZX-81.

A parte inferior da pilha está no RAMTOP (locação máxima da memória RAM em um microcomputador) e pode ocupar uma área tão grande da RAM quanto seja necessário. O indicador de pilha, ou registro SP, possui dezesseis bits; assim, ele armazena todo o endereço do último byte ocupado da pilha.

Os micros da linha Apple II, bem como o TK-2000, utilizam o microprocessador 6502.

Esse chip possui um único acumulador de oito bits e dois registros de indexação de oito bits cada: X e Y. Estes contêm deslocamentos *(off-sets)*, que são acrescentados a um endereço-base para se obter outro endereço. Esse método é muito utilizado para ler uma tabela de dados, por exemplo. O endereço-base, no caso, é o início da área de memória contendo uma tabela. Para ler a tabela, usamos um laço, que incrementa apenas o deslocamento a cada passagem. Com isso, o programa é direcionado para o próximo byte de tabela, cada vez que o laço é repetido.

É possível direcionar um programa em código de máquina para um endereço que contenha um outro endereço — desde que este se encontre na página zero da RAM, pois os endereços indiretos podem ter só oito bits de comprimento (isto limita em 255 o número de memórias endereçáveis por este método). Você pode deslocar um desses dois endereços, mas o registro X deve ser utilizado para deslocar a partir do primeiro endereço, e o registro Y deve deslocar a partir do segundo endereço.

O *registro processador* P de oito bits, também conhecido como *registro de estado (status-register)*, contém um sinalizador. Os sinalizadores de transporte C são os mais importantes e ocupam o bit 0; o sinalizador de 0 Z ocupa o bit 1; o sinalizador de excesso V ocupa o bit 6; e o sinalizador de sinal N ocupa o bit 7.

Existem mais três sinalizadores que você pode utilizar. O sinalizador de decimal D, que é o bit 3, é estabelecido quando o microprocessador efetua operações aritméticas em binário codificado em decimal (BCD). O sinalizador de pausa B e o sinalizador de interrupção I são utilizados em "rotinas de interrupção". Estas interrompem, a intervalos regulares, o fluxo normal dos seus programas em código de máquina para desempenharem funções vitais, como a varredura do teclado.

O 6502 também tem uma pilha que está limitada à página 1, com sua base em $01FF, pois o *indicador de pilha*, ou registro SP, tem apenas 8 bits. Esse indicador pode conter qualquer número de $00 a $FF e fornece o byte menos significativo do primeiro endereço livre localizado abaixo da pilha. O microprocessador já sabe que o bit mais significativo do endereço é 01, pois a pilha está na página 1.

O microprocessador existente nos computadores da linha TRS-Color é do tipo Motorola 6809E de oito bits, operando a uma freqüência de 0,895 MHz. Ele tem dois *acumuladores:* os registros A e B, que podem ser utilizados independentemente, como registros separados de oito bits, ou juntos, como um acumulador D de dezesseis bits.

Ele conta ainda com dois *registros de indexação* de dezesseis bits — X e Y — que retêm os endereços que podem ser aumentados ou deslocados, por meio da adição ou subtração de números (conhecidos como "deslocamentos" ou *off-sets)*, para fornecerem um outro endereço. Freqüentemente, essa forma de programação é utilizada para ler uma tabela de dados a partir de um endereço-base.

O *registro de código de condição* retém os diversos bits sinalizadores. O bit 0 é o sinalizador de transporte C; o bit 1 é o sinalizador de excesso V; o bit 2 é o sinalizador de zero Z; o bit 3 é o sinalizador de sinal N; o bit 5 é o sinalizador de meio transporte, que é ativado apenas com operações aritméticas com BCD (decimal codificado em binário, que é uma representação especial de números decimais); o bit 4 é o sinalizador da máscara de interrupção normal I; o bit 6 é o sinalizador da máscara de interrupção rápida F; e o bit 7 é o sinalizador de total E — todos são utilizados em rotinas de interrupção, que interrompem, a intervalos regulares, o fluxo normal do programa de código de máquina para desempenharem funções decisivas, como atualizar o cronômetro interno, fazer a varredura do teclado para ver se alguma tecla foi apertada etc.

O chip 6809 possui ainda um *registro de página direta* — ou DP — de oito bits. A função desse registro é armazenar o byte mais significativo dos endereços que você escolheu para a página direta, de modo que a locação nessa página possa ser endereçada apenas pelo seu byte menos significativo.

São dois os *indicadores de pilha*, de dezesseis bits cada: S e U. O primeiro revela o último endereço completo no hardware da pilha cuja base está em RAMTOP ou &H7FFF (a menos que tenha sido empurrado para baixo, de modo a deixar espaço a um programa em código de máquina). O indicador U, por sua vez, é utilizado com a pilha do usuário, cuja base deve ser normalmente definida pelo programador. A pilha do usuário é pouco utilizada; assim, o registro U serve como um registro extra de indexação de dezesseis bits pelos códigos de máquinas mais sofisticados no TRS-Color ou pelo programador de linguagem Assembly.

COMO DESENHAR EM BASIC

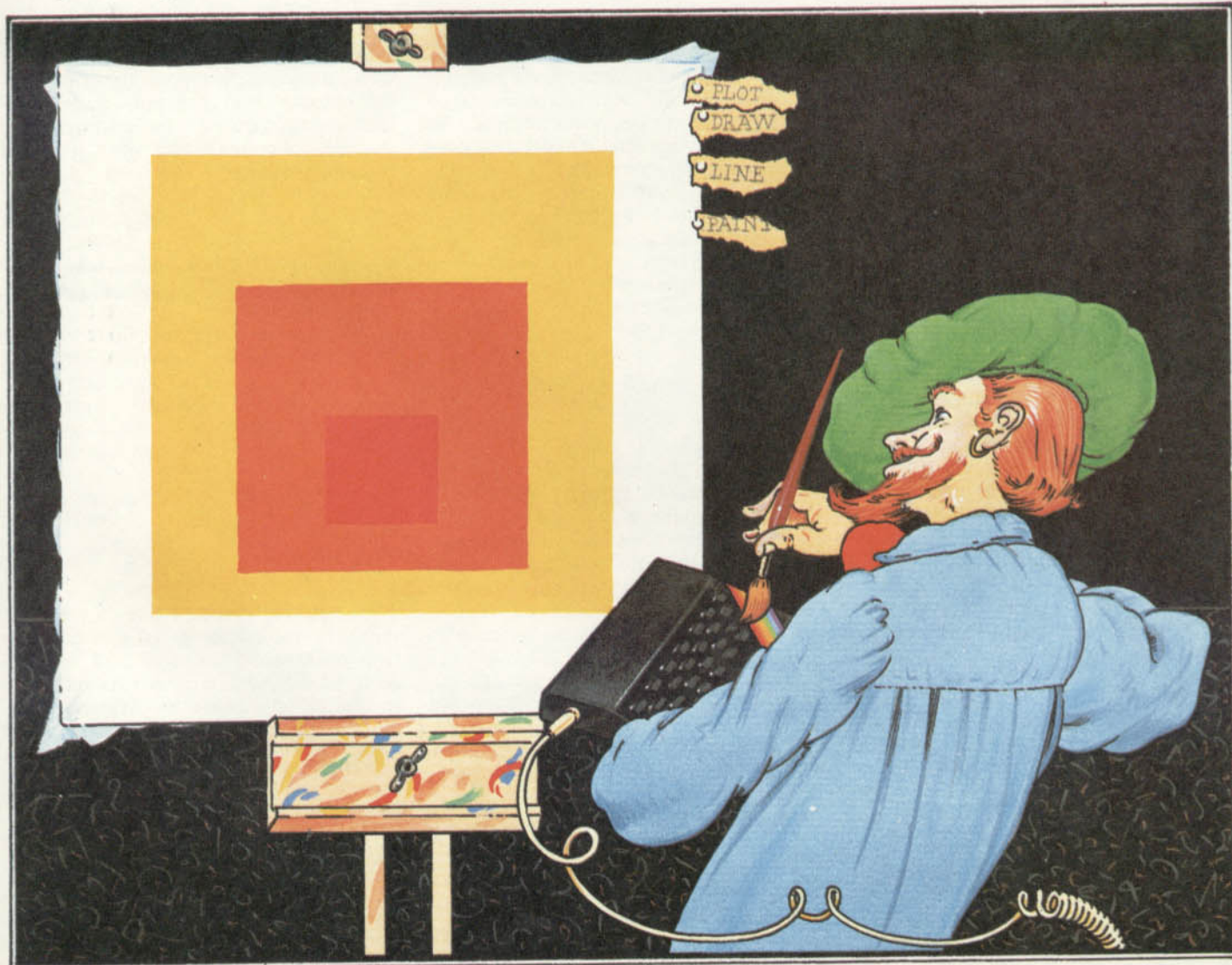
- UTILIZE OS COMANDOS GRÁFICOS DO BASIC
- DESENHE QUADRADOS, TRIÂNGULOS E CÍRCULOS
- USE **FLASH** PARA ANIMAR PROGRAMAS NO SPECTRUM
- GRÁFICOS NO APPLE E NO MSX

Alguns comandos simples e muita imaginação: isso é tudo que você precisa para produzir imagens sensacionais na tela do computador. Veja aqui por onde começar.

Ligue um aparelho de TV ou um monitor de vídeo ao seu computador e a telinha ficará à sua disposição. Praticamente todos os microcomputadores pessoais permitem que você desenhe nela, acrescente cor onde e como desejar, etc. Em pouco tempo você estará fazendo desenhos realmentte complicados.

Os gráficos para computador evoluíram muito nos últimos anos e atualmente são utilizados em muitos tipos de aplicações. Gráficos e tabelas constituem um exemplo óbvio, mas você também já deve ter notado o aumento no número de imagens geradas por computadores em propagandas de TV, revistas e até mesmo em efeitos especiais de filmes. Na verdade, muitas das ilustrações de INPUT foram produzidas ou modificadas por um computador.

É claro que os computadores utilizados para esses tipos de imagem sofisticada são muito maiores do que o modesto micro que você tem em casa. Mas existe uma quantidade impressionante de coisas que você pode fazer em sua maquininha doméstica.

A maioria dos micros entende comandos simples, tais como **PLOT**, **DRAW**, **PAINT** e **INK**. Para usá-los com eficiência e facilidade, contudo, é preciso compreender bem como a tela gráfica do computador é organizada.

Praticamente todos os computadores com capacidade gráfica de alta resolução utilizam a mesma forma de organização da tela. Esta é dividida em um conjunto de elementos individuais de imagem chamados *pixels* (termo que deriva da abreviatura da expressão inglesa *picture element*). Cada pixel é um pequeno retângulo na tela, que pode ser "ligado" ou "desligado" individualmente, por diferentes comandos. Os pixels são dispostos em uma espécie de matriz retangular, como um sistema de coordenadas, e cada um deles tem um "endereço", que geralmente é um par de números: sua posição na tela em relação ao eixo horizontal e ao vertical.

O número de pixels na tela define o seu *grau de resolução gráfica*. Em alguns computadores, você pode escolher entre desenhos de alta resolução com poucos "pixels" e linhas finas, ou desenhos de baixa resolução, com pixels maiores e linhas grossas. Muitos computadores permitem também que você selecione a cor que o pixel terá na tela ao ser iluminado.

Assim, desenhar na tela é apenas uma questão de dizer ao computador quais pixels devem ser acendidos. Em poucos minutos, experimentando os comandos gráficos mais simples, você estará fazendo belos desenhos sozinho.

Alguns micros, como os compatíveis com o ZX-81, têm uma capacidade gráfica relativamente limitada, pois seus programas exploram a utilização de dois comandos, apenas: o **PLOT**, que simplesmente acende um ponto na tela, e o **UNPLOT**, encarregado de apagá-lo. Além disso, a resolução gráfica da tela é bastante pequena. No entanto, esses micros podem ser equipados, opcionalmente, com uma placa ou um cartucho para gráficos de alta resolução.

Já os microcomputadores das linhas TRS-Color, Apple II, Sinclair Spectrum e MSX dispõem de recursos bastante sofisticados para desenhar em alta resolução.

Além disso, os micros mais modernos contam com comandos gráficos adicionais que permitem criar figuras geométricas, como círculos e retângulos, e pintá-las com cores diversas.

Para esses computadores, existe o conceito de *cursor gráfico*, ou seja, um ponto imaginário na tela que indica as coordenadas finais do último ponto ou reta traçados.

S

A imagem de alta resolução que o Spectrum produz na tela é de 256 pixels no sentido horizontal e 176 no sentido vertical.

Os pixels são numerados de uma maneira aparentemente estranha à primeira vista. Se você já utilizou as declarações **PRINT AT** antes, sabe que:

```
PRINT AT 10,15 ...
```

significa que o que deve ser impresso será colocado na tela em uma posição definida por 10 linhas, contadas a partir do alto da tela, e 15 colunas, contadas a partir da borda esquerda.

Entretanto, quando você emprega a instrução **PLOT** para acender um pixel, o primeiro parâmetro do comando fornece a distância a partir da margem esquerda do vídeo, e o segundo define a distância para cima, a partir do lado inferior da tela. Esse método, aparentemente em conflito com o anterior, segue a convenção adotada para o sistema de coordenadas ortogonais (cartesianas). Assim, por exemplo, o comando

```
PLOT 10,15
```

diz ao computador para colocar um sinal na tela distante 10 pixels da margem esquerda e 15 linhas contadas de baixo para cima. Equivale, portanto, a $X = 10$ e $Y = 15$. Se você fizer a experiência verá o ponto aparecer na parte inferior esquerda da tela. As posições horizontais são numeradas de 0 a 255, e as verticais, de 0 a 175. Para sentir melhor como funciona a numeração das posições de tela, tente digitar essas linhas, uma de cada vez:

```
PLOT 0,0
PLOT 0,175
PLOT 255,0
PLOT 255,175
```

Os quatro pontos vistos agora na tela marcam os limites da área na qual se pode desenhar alguma coisa. Tudo o que estiver fora desses limites cairá numa parte da tela conhecida como moldura ou borda (**BORDER**) (veja mais adiante). Você pode usar, neste caso, o comando **PLOT** com todas as cores disponíveis no Spectrum.

COMO DESENHAR RETAS

O comando **DRAW** do Spectrum faz exatamente o que você espera: ele desenha uma linha reta.

Antes, porém, o computador tem que saber onde começar a desenhá-la, qual será o seu comprimento, e em que direção você quer orientá-la.

Para iniciar a reta, você pode utilizar uma posição qualquer na tela, conforme foi definido para o comando **PLOT**. Tente essas retas em ordem, sem limpar a tela entre elas:

```
PLOT 100,75
DRAW 50,0
```

Se você não fornecer uma nova coordenada na tela, por intermédio do comando **PLOT** (que pula para ela, sem desenhar nada), o computador continuará a desenhar a partir da última posição, ou seja, do ponto final da última reta traçada (que é onde está o cursor gráfico). Para comprovar isso, adicione as seguintes retas, uma de cada vez, às duas que você acabou de desenhar:

```
DRAW 0,50
DRAW -50,0
DRAW 0,50
```

Como você pode observar, é necessário empregar números positivos para desenhar para cima ou para a direita e números negativos para desenhar para baixo ou para a esquerda.

Para traçar uma reta inclinada, você deve seguir a mesma regra: primeiro, estabeleça quantas posições para a direita ou para a esquerda o cursor gráfico deve se movimentar, depois quantas posições para cima ou para baixo. Por exemplo, acrescente essa reta àquelas já traçadas acima:

```
DRAW 50,50
```

CONSTRUA UM PROGRAMA

Eis aqui um programa mais interessante, utilizando o comando **DRAW**. Para que você possa ver o que está acontecendo, entre e execute as linhas dos programas abaixo:

```
10 INK 2
20 PLOT 0,175
30 LET t=255: LET r=-175:
LET b=-255: LET l=169
40 DRAW t,0: DRAW 0,r: DRAW b
,0: DRAW 0,l
```

Se você digitou o programa corretamente (e não cometeu erros, como utilizar o número 1 no lugar do L minúsculo, ou vice-versa), obterá um retângulo sem um dos lados. Em seguida, adicione ao programa as linhas abaixo:

```
50 LET t=t-6
60 LET r=r+12: LET b=b+12:
LET l=l-12
70 DRAW t,0: DRAW 0,r: DRAW b
,0: DRAW 0,l
```

Ao executar o programa, você notará que agora apareceu um outro retângulo menor, dentro do primeiro. Acrescente então o resto do programa:

```
80 LET t=t-6
90 GOTO 50
```

Veja o que acontece: cada vez que o seu programa executa o laço entre as linhas 50 e 90, deduz 12 de cada um dos números positivos. A variável **t** (topo), por exemplo, diminui 255 para 243, depois para 231... para 219... e assim por diante. Eventualmente, ele vai aquém do zero e se torna um número negativo — assim, a reta que está sendo controlada pelo laço muda de direção. Do mesmo modo, as variáveis inicialmente negativas (tais como **b**, para baixo) podem tornar-se positivas e também mudar de direção.

Para observar mais de perto esse processo, rode novamente o programa com a seguinte adição:

```
85 PAUSE 100
```

LUZ E SOMBRA

O Spectrum não tem a declaração **COLOR**, usada para colorir fácil e rapidamente uma área de fundo.

Todavia, você pode "sombrear" o desenho, utilizando uma série de retas paralelas de uma só cor. Mas, como cada nova reta precisa ter seu próprio comando **PLOT**, os programas tendem a se tornar demasiado lentos.

Um modo de se contornar essa dificuldade é utilizar um laço **FOR...NEXT:** o programa seguinte desenha a parede direita de uma casa (figura 2):

```
20 FOR h=56 TO 100 STEP 3
30 PLOT 160,h
40 DRAW 23,12
50 NEXT h
60 PLOT 183,110 : DRAW 0,-9/
 : DRAW    -23,-13: DRAW 0,100
```

Como isso funciona? Cada vez que o laço é executado, o programa desenha uma linha de 23 pixels, inclinando-a de 12 pixels, da esquerda para a direita. A cada nova linha, ele inicia 160 pixels à esquerda.

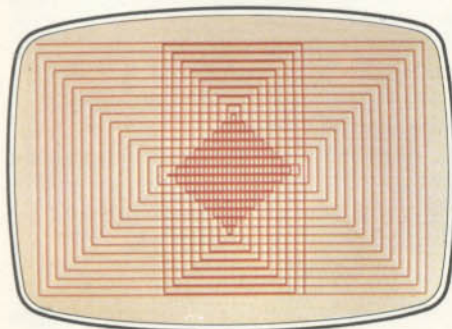



Fig. 1: Um padrão de gráfico para o Spectrum.

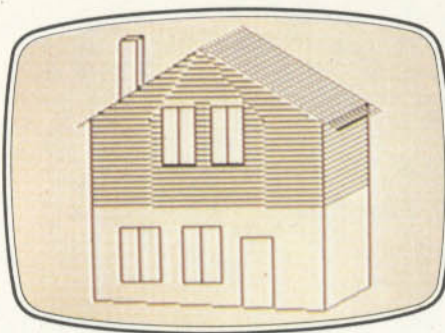



Fig. 2: A casa em perspectiva.

Mas a cada "viagem" do laço a linha 20 acrescenta 3 à variável **h**, que representa a altura da posição **PLOT**. Assim, o início de cada linha será três pixels mais alto que o da anterior.

Acrescente as próximas linhas e sua casa passará a ter a maior parte do telhado:

```
 70 LET r=100
 80 FOR p=140 TO 150
 90 PLOT r,p
100 DRAW 69,-44
110 LET r=r+2
120 NEXT p
```

Existem aqui duas variáveis, **p** e **r**. Isso se deve a que as posições **PLOT** movimentam-se não apenas para cima mas também para a direita.

Observe a utilização de números negativos na linha 100 para fazer o declive do telhado para baixo. As próximas linhas completam o contorno da casa:

```
130 PLOT 100,140
140 DRAW -60,-40
150 PLOT 46,103
160 DRAW 0,-96
170 DRAW 113,-7
```

Agora você pode terminar a casa, uma porta, as janelas e uma chaminé com o auxílio dos comandos **PLOT** e **DRAW**. Se você decidir copiar a casa que aparece na figura 1, eis aqui algumas sugestões:

O triângulo do revestimento abaixo do telhado requer três variáveis. A primeira define a altura onde cada linha começa. A segunda controla a distância, à direita, de onde ela parte. A terceira, finalmente, determina o comprimento da linha.

As janelas do andar superior são traçadas sobre o revestimento, depois que este é desenhado. Para fazer isso, você precisará jogar espaços em branco, usando o **PRINT AT**, nos pontos adequados. Em seguida, trace o contorno das janelas, utilizando **PLOT** e **DRAW** da maneira usual.

COMO DESENHAR CÍRCULOS

A declaração **CIRCLE** do Spectrum também se comporta de modo previsível: ela desenha um círculo.

É necessário não esquecer que os primeiros dois números após uma declaração **CIRCLE** definem as coordenadas do centro do círculo — o qual não é traçado —, enquanto o terceiro número fornece o seu raio em pixels. Assim a linha:

```
CIRCLE 127,87,50
```

... desenha um círculo de 100 pixels de diâmetro (50 de raio) no meio da tela (posição x = 127 e y = 87).

No programa da casa, por exemplo, você pode "instalar" uma janela redonda na porta da frente, utilizando a declaração **CIRCLE**.

Se você seguir um **CIRCLE** com uma declaração **DRAW**, a linha que você desenhar começará a partir do ponto onde o círculo termina.

CORES NO SPECTRUM

O Spectrum tem uma gama de oito cores, identificadas por rótulos acima das teclas numéricas de 0 a 7, que são utilizadas para chamá-las quando isso se faz necessário. O comando **BORDER** (bordas) controla a área externa da tela, sobre a qual você não pode desenhar nem imprimir nada.

Os outros comandos de cores podem ser utilizados de duas maneiras bem diversas. Se você rodar um programa com, por exemplo,

```
10 BORDER 2:PAPER 2:INK 7
```

tudo o que se seguir no programa, depois que essa linha for executada, será impresso em branco **(INK 7)**, sobre uma tela de fundo vermelho **(PAPER 2)**. Tal situação só será alterada quando você substituir essa linha por outra que modifique os comandos de cor. Note que a moldura é da mesma cor que o fundo da tela **(BORDER 2)**, mas você poderia ter especificado outra cor para ela. Até mesmo as listagens dos programas aparecerão nas cores indicadas, o que pode dificultar a sua leitura. Se, por outro lado, você iniciar com:

```
10 PRINT PAPER 2:INK 7:AT
    15,10; "*"
```

então, apenas a pequena área da tela onde o asterisco entre aspas aparece ficará vermelha.

Observe que no primeiro exemplo as declarações de cor são seguidas por dois pontos, ao passo que no segundo exem-

plo cada uma delas é separada por um ponto e vírgula.

Você pode incorporar essas declarações de cor a qualquer linha do programa, utilizando **PLOT**, **DRAW** ou **CIRCLE**. Tente essas linhas limpando a tela (comando **CLS**) entre as linhas:

```
PLOT PAPER 1:INK 7:125,87
```

(Dá para ver o ponto?)

```
PLOT 125,87:DRAW INK 2;50,50
PLOT 125,87:DRAW PAPER 1; INK
     7;50,50
CIRCLE INK 4;127,87,50
```

e até mesmo esta:

```
CIRCLE PAPER 2;INK 6;125,87,50
:DRAW
PAPER 3;INK 7;75,0
```

Esta última combinação pode ser utilizada para produzir alguns efeitos espetaculares.

O último dos comandos gráficos do Spectrum tratados neste artigo — o **FLASH** — é igualmente espetacular, e muito utilizado em programação de jogos. O que ele faz é reverter rapidamente as cores da tinta e do papel.

O **FLASH** deve ser seguido por um número — tanto o 1 para ligar, como o 0 para desligar; eis aqui um exemplo de sua utilização:

```
10 PAUSE 0
20 IF INKEY$="" THEN  GOTO 30
30 FOR n=10 TO 23: SOUND .015
,n: NEXT n
40 BORDER 2: PAPER 2: INK 6:
FLASH 1
50 PRINT "NAO TOQUE !"
60 PRINT "ESTE COMPUTADOR E P
ERIGOSO!"
70 PRINT
80 GOTO 30
```

Quando você tiver entrado esse programa, rode-o. Então pressione qualquer tecla... e saia de perto!

O ZX-81 utiliza o comando **PLOT** de um modo parecido ao do Spectrum: **PLOT**, seguido por dois números que controlam a posição do pixel. Mas existe uma diferença: os pixels do ZX-81 são dezesseis vezes maiores do que os pixels do Spectrum, pois neste existem apenas quatro pixèls em cada espaço de caractere, enquanto que o Spectrum tem 64 pixels em um espaço de caractere.

Como resultado, você não conseguirá fazer desenhos ou padrões tão detalhados; mas tornará cada pixel mais visível.

O programa a seguir traça um padrão gráfico na tela. Como ele recomeça cada vez que vai para a linha 70, você deverá pressionar a tecla <**BREAK**> quando quiser parar:

```
10 LET X=INT (RND*32)
20 LET Y=INT (RND*24)
30 PLOT X,Y
40 PLOT 63-X,Y
50 PLOT X,43-Y
60 PLOT 63-X,43-Y
70 GOTO 10
```

O ZX-81 não tem o comando **DRAW**, mas você pode utilizar a instrução **PLOT** para compor uma reta com pontos. Você poderá ver como isso funciona com o programa seguinte, o qual desenha um certo número de quadrados utilizando dois laços **FOR...NEXT**.

```
10 FOR N=0 TO 20 STEP 2
20 FOR M=N TO 43-N
30 PLOT M,N
40 PLOT M,43-N
50 PLOT N,M
60 PLOT 43-N,M
70 NEXT M
80 NEXT N
```

E de um modo parecido, você pode imitar o comando **CIRCLE**, empregando a seguinte rotina:

```
10 FOR N=0 TO 2*PI STEP .1
20 PLOT 32+20*SIN N,22+20*COS N

30 NEXT N
```

Ela utiliza as duas funções matemáticas — o seno (**SIN**) e o cosseno (**COS**) — para calcular as coordenadas dos pixels que compõem o círculo. Elas serão explicadas em detalhes em um artigo posterior, mas, no momento, você pode experimentar com o tamanho do círculo, modificando os dois números 20. O número máximo que você pode ter é 22 (maior que isso não caberá na tela). Outra maneira é modificar a posição do círculo. No momento, seu centro está em 32,22 (você pode ver essas coordenadas na linha 20). Não se esqueça de fazer o círculo menor se você movê-lo para perto da borda da tela.

Os microcomputadores compatíveis com Apple II também têm recursos razoáveis para a elaboração de desenhos em alta resolução, em cores.

O Apple tem três modalidades de tela: a tela de texto (invocada pelo comando **TEXT** e limpada pelo comando **HOME**); a tela gráfica de baixa resolução (que tem 40 pixels na direção horizontal e 40 pixels na direção vertical, usando dezesseis cores); e a tela gráfica de alta resolução (que tem 280 pixels na direção horizontal e 192 na direção vertical, com seis cores). O comando para ligar a tela gráfica de baixa resolução é o **GR**; e, para a tela de alta resolução, o **HGR**.

Existe um conjunto diferente, embora parecido, de comandos gráficos para os modos de alta e de baixa resolução. Para o modo de baixa resolução temos os seguintes: **COLOR**, **PLOT**, **HLIN** e **VLIN**. Para a alta resolução, os comandos que mostraremos nesta lição são o **HCOLOR** e o **HPLOT**.

Além disso, existem outros comandos mais sofisticados para o modo de alta resolução, tais como **DRAW**, **ROTATE**, **SCALE**, etc., que serão tratados em uma lição futura.

Os micros da linha Apple II têm a limitação de não permitir a exibição simultânea de letras e de gráficos de baixa e de alta resolução na mesma tela. Só o modelo nacional TK-2000, da Microdigital, tem essa capacidade.

LINHAS RETAS E COR

Para demonstrar a utilização dos comandos gráficos de alta resolução no Apple II e no TK-2000, vamos fazer um desenho bem simples, passo a passo: um aparelho de TV em que aparecem uma série de pontos coloridos, distribuídos aleatoriamente na tela. O programa começa com as linhas a seguir, que desenham a caixa do televisor:

```
10   HGR2
20   HCOLOR= 3
30   HPLOT 10,75 TO 200,75
40   HPLOT 200,75 TO 200,190
50   HPLOT  TO 10,190
60   HPLOT  TO 10,75
```

A linha 10 define o tipo de tela gráfica a ser utilizada, com o comando **HGR2**. O **HGR** significa que a tela será de alta resolução (*High-resolution GRaphics*), e o número 2 diz ao computador que a tela será inteiramente tomada pelos gráficos (se não houvesse esse 2, o computador reservaria automaticamente cinco linhas de texto na parte de baixo da tela). Sem esse comando, o computador não aceitaria os comandos gráficos subseqüentes.

A linha 20 define a cor a ser usada para o traço em alta resolução. É usado o comando **HCOLOR**, seguido do número da cor desejada. O Apple não tem um comando para definir a cor de fundo, como é o caso do MSX e do TRS-Color. Se for necessário ter uma cor de fundo diferente do preto (a normal da tela), usa-se uma sucessão de li-

nhas paralelas da mesma cor (essa manobra, porém, leva muito tempo).

As linhas 30 a 60 traçam um retângulo na tela, com o canto superior esquerdo em (10,75) e o canto inferior direito em (200,190). O comando utilizado é o **HPLOT**. Esse comando pode ser usado de várias maneiras.

A variante do **HPLOT** mostrada nesse segmento de programa serve para traçar retas. Os parâmetros básicos para o **HPLOT** são conjuntos de dois números ou expressões numéricas, separados por uma vírgula. Tais parâmetros indicam a posição em que o cursor gráfico deverá ser colocado na tela de alta resolução. Por exemplo: **HPLOT 10,75...** diz ao computador para começar a traçar uma reta em alta resolução, nas coordenadas X = 10 e Y = 75 (ou seja, 10 pixels distantes da margem esquerda da tela, e 75 pixels abaixo do topo).

Para traçar uma reta, usa-se o comando do **HPLOT** seguido da palavra **TO**. Por exemplo, na linha 30 do programa, traçamos uma reta começando no ponto (10,75), e indo até o ponto (200,75) — portanto, paralela ao eixo horizontal — usando o comando:

```
HPLOT 10,75 TO 200,75
```

O cursor gráfico agora fica no último ponto da tela depois do **TO:** no exemplo acima, em (200,75). Se quisermos traçar uma reta a partir daí, não é necessário especificar um novo começo para ela. Basta usar o comando assim:

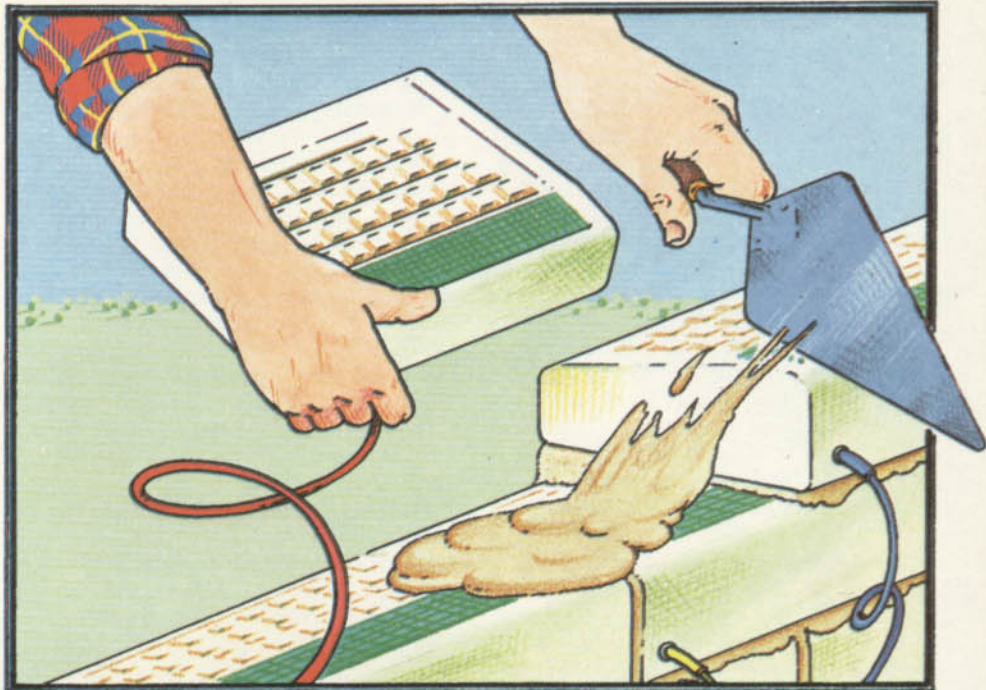
```
HPLOT TO 200,190
```

Essa nova forma vai traçar uma linha de (200,75) a (200,190) — uma reta vertical, portanto.

A linha 300 simplesmente "congela" a tela gráfica, impedindo que o fim do programa a cancele.

Vamos traçar agora o restante do televisor, adicionando as linhas abaixo:

```
70  HPLOT 25,85 TO 155,85 TO 15
5,180 TO 25,180 TO 25,85
80  HPLOT 10,75 TO 45,50
90  HPLOT 200,75 TO 235,50
100  HPLOT 200,190 TO 235,165
110  HPLOT 45,50 TO 235,50 TO 2
35,165
120  HPLOT 190,60 TO 230,0
130  HPLOT 190,60 TO 150,0
140  HPLOT 175,85 TO 179,85 TO
179,89 TO 175,89
150  HPLOT 175,105 TO 179,105 T
O 179,109 TO 175,109
160  HPLOT 175,135 TO 179,135 T
O 179,139 TO 175,139
170  HPLOT 175,155 TO 179,155 T
O 179,159 TO 175,159
```

A linha 70 traça um outro retângulo dentro do primeiro. Note que podemos

utilizar uma série de cláusulas **TO**, dentro da mesma linha com o **HPLOT**. A linha 70 funciona de modo idêntico às quatro linhas dos números 30 a 60.

As linhas 80 e 90 servem para desenhar a antena. Note que linhas inclinadas são traçadas exatamente com a mesma técnica do **HPLOT**. As linhas restantes, de 100 a 170, servem para traçar o restante do desenho do televisor, e dar perspectiva a ele.

Finalmente, para fazer aparecer um "chuvisco" colorido na tela do televisor que está sendo esboçado no vídeo do seu computador, adicione as linhas:

```
180 X =   INT ( RND (1) * 130) +
 26
190 Y =   INT ( RND (1) * 95) +
86
200  COLOR=  RND (1) * 10
210  HPLOT X,Y
220  GOTO 180
```

As coordenadas dos pontos aleatórios são definidas nas linhas 180 e 190 e são armazenadas nas variáveis X e Y. Note que a expressão aritmética sorteia, usando a função **RND**, um número inteiro dentro dos limites da tela.

O **HPLOT** mostrado aqui é mais simples, pois é seguido apenas por dois números ou duas expressões numéricas, separadas entre si por uma vírgula. Esses dois parâmetros indicam a posição em que o cursor gráfico deverá ser colocado na tela de alta resolução.

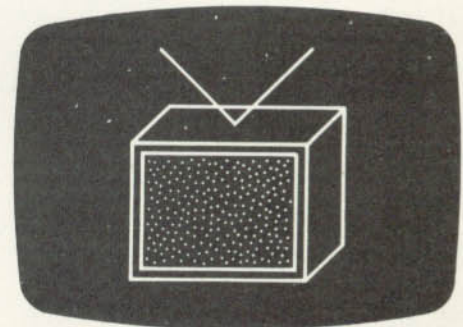
```
HPLOT X,Y
```

diz ao computador para traçar um ponto em alta resolução, nas coordenadas X e Y.

Finalmente, a linha 220 gera novos pontos aleatórios em um laço infinito. Para interrompê-lo, use as teclas **<CTRL> <C>**.

CÍRCULOS NA TELA

Os micros da linha Apple não têm um comando específico para traçar círculos, mas você pode imitar o comando **CIRCLE** utilizando a seguinte rotina:

```
10   HOME : INPUT "COR:";C
12   INPUT "CENTRO (X,Y):";XC,YC
14   INPUT "RAIO:";R
15   HCOLOR= C
16   HGR
17   HPLOT XC,YC + R
20   FOR N = 0 TO 6.2856 STEP .1
```

Formas simples como esta televisão podem ser desenhadas no Apple II e no TK-2000.

```
30  HPLOT  TO XC + R *  SIN (N)
,YC + R *  COS (N)
40  NEXT N
```

Essa rotina utiliza as duas funções matemáticas — o seno (**SIN**) e o cosseno (**COS**) — para calcular as coordenadas de cada pixel que compõe o círculo. Tais funções serão melhor explicadas em uma futura lição; no momento, porém, você pode experimentar com o tamanho do círculo, modificando os dois números 20 na linha 20. O número máximo que se pode ter é 22 (a tela não comporta um valor mais alto). Você pode também modificar a posição do círculo. Seu centro está em 32,22; observe essas coordenadas na linha 20. Não se esqueça de fazer o círculo menor se você movê-lo para perto da borda da tela.

T

Assim que você liga o computador, ele exibe a tela de texto. Nela você pode mostrar todos os caracteres do teclado, mais os caracteres gráficos da ROM, já utilizados antes.

Se você quiser desenhar qualquer coisa mais sofisticada do que aquilo que esses simples gráficos permitem, deve empregar os gráficos de alta resolução da máquina. Estes são obtidos em uma tela completamente diferente da tela de texto — pois, nos computadores compatíveis com o TRS-Color, não é possível colocar-se, ao mesmo tempo, texto e gráficos de alta resolução na mesma tela.

RETAS CRUZADAS

Eis aqui um programa que desenha duas retas cruzadas na tela, utilizando a tela gráfica em modo 3:

```
20 PMODE 3,1
30 PCLS
40 SCREEN 1,0
50 LINE (0,191)-(255,0),PSET
60 LINE (0,0)-(255,191),PSET
70 GOTO 70
```

O programa começa especificando a modalidade gráfica e a página (parte da memória do computador) em que você deseja colocar a informação gráfica. A linha 20 declara isso com **PMODE 3,1**. Essa informação diz à máquina para desenhar em modalidade de gráficos 3 e para armazenar qualquer coisa que você desenhar na tela na página de memória gráfica número 1. Se você quiser desenhar apenas uma tela cheia de gráficos será mais fácil especificar a página 1.

A linha 30 limpa a tela de alta resolução. **PCLS** é o comando para alta resolução, equivalente ao **CLS**.

Para "ligar" a tela de alta resolução o programa utiliza o comando **SCREEN 1,0**. O conjunto de cores é escolhido colocando-se 0 como o segundo no comando **SCREEN**: esse conjunto consiste de verde, amarelo, azul e vermelho. Se, ao invés disso, você tivesse colocado 1, o conjunto de cores seria branco, azul ciano, magenta e laranja.

Esse programa desenha linhas em verde e vermelho, que são as cores pré-definidas (são as normalmente utilizadas pelo computador, se você não especificar nenhuma outra). Para utilizar qualquer outra combinação do conjunto de cores você deverá utilizar o comando **COLOR**. Isso será explicado em detalhes, futuramente.

A primeira reta é desenhada pelo comando **LINE** na linha 50. Os números entre parênteses dizem ao computador em que pontos você quer que a linha comece e termine. A tela de alta resolução é dividida em 256 pixels por 192 — no sentido vertical, eles são numerados de 0 a 191, a partir do alto; no sentido horizontal, de 0 a 255, a partir da esquerda. Note que a convenção adotada no TRS-Color para as coordenadas cartesianas é o oposto ao normal.

Portanto, **LINE (0,191)-(255,0)** traça uma reta com a origem em X=0 e Y=191 e fim em X=255 e Y=0.

Finalmente, **PSET** sozinho, no comando **LINE**, diz à máquina para desenhar a cor de maior número do conjunto de cores — neste caso, o vermelho.

A linha 60 desenha a próxima reta. Isto funciona exatamente do mesmo modo como na linha 50, mas os pontos iniciais e finais são definidos novamente.

Para evitar que o computador retorne automaticamente para a tela de texto, uma vez que a tela de alta resolução tenha sido desenhada, estabelecemos um laço sem fim: é o que a linha 70 faz, com o comando **GOTO 70**. Se o computador ligasse novamente a tela de texto, não seria possível ver o que foi desenhado na tela de alta resolução.

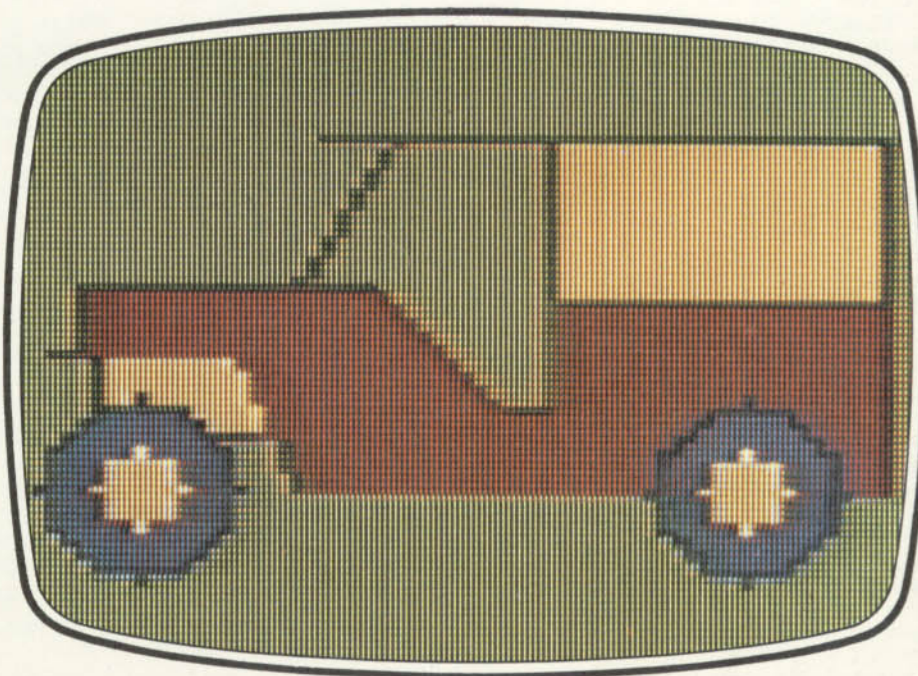
CÍRCULOS NO TRS-COLOR

É muito fácil desenhar círculos no TRS-Color, pois o seu BASIC possui um comando **CIRCLE**.

Este programa desenhará três círculos com tamanhos diferentes, cada um com uma cor diferente.

```
20 PMODE 3,1
30 PCLS
40 SCREEN 1,0
50 CIRCLE (70,95),10,2
60 CIRCLE (118,95),20,3
70 CIRCLE (184,95),30,4
80 GOTO 80
```

A modalidade gráfica é escolhida como no programa anterior, assim como o conjunto de cores para os círculos.

O comando **CIRCLE** nas linhas 50 a 70 tem quatro elementos. O primeiro e

O desenho do jipe torna-se fácil com o comando *PAINT* do TRS-Color.

o segundo números definem as coordenadas do centro do círculo, o terceiro é o seu raio, em pixels, e o quarto é a cor. Assim, o círculo desenhado pela linha 50 tem o seu centro em (70,95), um raio de 10 pixels e está colorido de amarelo (Cor 2).

O círculo desenhado pela linha 60 é azul, com raio de 20 pixels, e com centro em (118,95). O último círculo, desenhado pela linha 70, é vermelho, com raio de 30 pixels e com centro em (184,95).

Finalmente, a linha 80 é um laço que impede que o programa pare e desligue a tela de alta resolução.

DESENHE UM JIPE

Agora você possui os ingredientes para desenhar muitas coisas diferentes. Digite e rode o programa abaixo e você verá o esboço da carroçaria de um jipe, ainda sem os detalhes internos mostrados na figura 3.

```
20 PMODE 3,1
30 PCLS
40 SCREEN 1,0
50 LINE (108,64)-(188,64),PSET
60 LINE-(188,116),PSET
70 LINE-(104,116),PSET
80 LINE-(96,96),PSET
90 LINE-(70,96),PSET
110 LINE-(74,96),PSET
120 LINE-(74,86),PSET
130 LINE-(114,86),PSET
140 LINE-(132,104),PSET
150 LINE-(140,104),PSET
160 LINE-(140,64),PSET
170 LINE(76,96)-(76,108),PSET
180 LINE-(100,108),PSET
320 GOTO 320
```

Assim como antes, a modalidade escolhida é a de número 3. A linha 50 começa traçando uma linha vermelha de (108,64) a(188,64). Para continuar o desenho a partir do mesmo ponto, não se deve dizer ao computador por onde começar — ele apenas prossegue de onde estiver no momento. As linhas 60 a 160, portanto, apenas definem os finais de linhas sucessivas.

A linha 170, no entanto, desenha uma linha a partir de um novo ponto inicial; assim tanto o início como o fim devem ser definidos.

Acrescente agora algumas linhas para definir o pára-brisa, as rodas e as calotas:

```
200 LINE(140,88)-(188,88),PSET
220 LINE(118,64)-(104,86),PSET
230 CIRCLE(82,116),14,3
250 CIRCLE(82,116),6,2
270 CIRCLE(168,116),14,3
290 CIRCLE(168,116),6,2
```

A linha 200 desenha a janela, enquanto o pára-brisa é determinado pela linha 220. O comando **CIRCLE** é utilizado pelas linhas 230 e 270 para definir as rodas e pelas linhas 250 e 290 para desenhar as calotas.

PINTE UMA ÁREA

Agora que o desenho está feito, é fácil colorir o seu jipe: você nem precisa de uma pistola de pintura, porque a máquina faz isso através do comando **PAINT**.

O comando **PAINT** preenche com uma cor específica qualquer forma que você tenha desenhado na tela. Ele também possui quatro elementos. Os dois primeiros, como você já deve ter adivinhado, marcam o local onde a pintura deve começar. O terceiro é o código da cor a ser utilizada. E o quarto é o código de cor da borda, ou da linha onde a pintura deve acabar.

Acrescente essas linhas e você verá, na prática, como fica o programa:

```
190 PAINT(90,100),2,4
210 PAINT(120,110),4,4
240 PAINT(82,116),3,3
260 PAINT(82,116),2,2
280 PAINT(168,116),3,3
300 PAINT(168,116),2,2
310 PAINT(180,80),2,4
```

Rode agora o programa: como num passe de mágica, o jipe se cobrirá de cores.

Não é possível colocar ao final do programa todas as linhas com **PAINT**, por uma razão muito importante:

A área que você quer colorir deve ser fechada completamente por uma série de retas da mesma cor (a cor da borda), ou pela borda da tela. Qualquer linha de cor diferente que estiver localizada no meio da figura geométrica a ser preenchida será pintada por cima.

As cores disponíveis no **PMODE 3,1** são: verde (1), amarelo (2), azul (3) e vermelho (4). Se você quiser pintar o jipe de modo diferente, tudo o que tem a fazer é modificar o número de cor da pintura nas linhas **PAINT** correspondentes.

Você pode, se quiser, acrescentar outras partes ao jipe, equipando-o com novos elementos, como uma roda de direção, por exemplo. Para isso, empregue os comandos **LINE** e **CIRCLE**.

MSX

Assim que você ligar o computador, ele exibirá a tela de texto, na qual podem ser mostrados todos os caracteres do teclado, mais os caracteres do gráfico da ROM. Estes últimos já foram utilizados em programas anteriores.

Para desenhar figuras mais sofisticadas do que aquelas que vimos até agora, é necessário utilizar os gráficos de alta resolução da máquina. Estes são obtidos em uma tela completamente diferente da tela de texto, já que os computadores da linha MSX não permitem que sejam colocados, ao mesmo tempo, texto e gráficos de alta resolução numa única tela.

VOLTANDO À LINHA RETA

Eis aqui um programa que desenha duas retas cruzadas no vídeo, utilizando a tela gráfica em modo 2:

```
10 SCREEN 2
20 LINE(0,191)-(255,0),15
30 LINE(0,0)-(255,191),1
40 GOTO40
```

O programa começa especificando a modalidade gráfica e a página (parte da memória do computador) na qual você deseja colocar a informação gráfica. A linha 20 declara isso com **SCREEN 2**, dizendo à máquina para desenhar em modalidade de gráficos 2.

A primeira reta é desenhada pelo comando **LINE**, na linha 20. Os números entre os parênteses dizem ao computador onde a linha deve começar e terminar. A tela de alta resolução é dividida em 256 pixels por 192: no sentido vertical eles são numerados de 0 a 191, a partir do alto; no sentido horizontal, de 0 a 255, a partir da esquerda.

Portanto, **LINE (0,191)-(255,0)** traça uma reta com origem em $X=0$ e $Y=191$ e fim em $X=255$ e $Y=0$.

O número no final do comando **LINE** diz à máquina que cor ela deve empregar (neste caso, existem dezesseis cores, numeradas de 0 a 15).

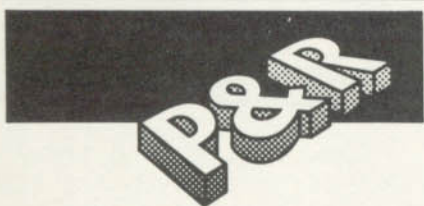
A linha 30 desenha a próxima reta. Isso funciona do mesmo modo que na linha 20, mas os pontos iniciais e finais são definidos novamente.

Para evitar que o micro retorne à tela de texto uma vez que a tela de alta resolução tenha sido desenhada, estabelecemos um laço sem fim: é o que a linha 40 faz, com **GOTO 40**. Se o computador religasse a tela de texto, você não seria capaz de ver o que foi desenhado na tela de alta resolução.

CÍRCULOS NO MSX

Assim como no TRS-Color, é fácil desenhar círculos no MSX, pois o seu

BASIC conta com um comando chamado **CIRCLE**. Este programa desenha três círculos com tamanhos diferentes, cada um de uma cor:

```
10 COLOR 4,15
20 SCREEN2
30 CIRCLE(70,95),10,2
40 CIRCLE(118,95),20,4
50 CIRCLE(184,95),30,8
60 GOTO 60
```

Qual a diferença entre gráficos de baixa, média e alta resolução?

O grau de resolução gráfica a ser obtido em um computador depende do número de pixels que se pode endereçar individualmente na tela. O pixel (que vem da abreviação, em inglês, de *picture element*) é o pontinho gráfico que um comando BASIC acende ou apaga. Naturalmente, como a tela tem um tamanho limitado, a resolução é proporcional às dimensões físicas de cada pixel.

Do ponto de vista técnico, a alta resolução gráfica é obtida quando há mais de 100 000 pixels por tela (por exemplo, o micro de tipo IBM-PC tem 600 pixels na horizontal por 200 na vertical). A resolução média ficaria com algo entre 30 000 e 100 000 pixels; a essa faixa pertencem o Sinclair Spectrum, o TRS-Color, o Apple II e o TK-2000. Já a resolução baixa tem de 2 500 a 10 000 pixels por tela (por exemplo, o TRS-80 e o ZX-81). Entretanto, muitos fabricantes (como os da linha Apple II) chamam de alta resolução os gráficos tecnicamente definidos como de média resolução.

O TRS-80 pode ser usado para desenhar em BASIC?

Sim, embora os seus gráficos apareçam apenas em baixa resolução (128 por 48 pixels, em preto e branco). O comando a ser utilizado para "acender" um pixel na tela é o **SET**. Esse comando deve ser seguido de dois parâmetros ou expressões numéricas: o primeiro **(X)** indica a posição horizontal do pixel e o segundo **(Y)**, a sua posição vertical. Ele equivale, portanto, ao comando **PLOT** dos micros da linha Sinclair. Existem ainda os comandos gráficos **RESET** (para "apagar" o pixel aceso) e **POINT** (uma função que informa se um determinado pixel está aceso ou apagado).

A modalidade gráfica é escolhida como no programa anterior. Só que a cor de fundo da tela é definida antes, com o comando **COLOR**. O primeiro número é a cor do caractere (cor de frente), e o segundo, a cor de fundo.

O comando **CIRCLE** nas linhas 30 a 50 tem quatro elementos. O primeiro e o segundo números definem as coordenadas do centro do círculo, o terceiro é o seu raio, em pixels, e o quarto é a cor. Assim, o círculo desenhado pela linha 30 tem o seu centro em (70,95), um raio de 10 pixels e é colorido de amarelo (cor 2).

O círculo desenhado pela linha 40 é azul, com um raio de 20 pixels, e centro em (118,95). O último círculo, desenhado pela linha 50, é vermelho, com raio de 30 pixels e centro em (184,95). A linha 60 é um laço que impede que o programa pare e desligue a tela de alta resolução.

DESENHE O JIPE NO MSX

Como você já deve ter percebido, algumas seções de nosso texto repetem seções anteriores. Isso se deve à similaridade dos procedimentos nos diversos micros. Agora, digite e rode o programa abaixo.

```
10 COLOR 4,15
20 SCREEN2
30 LINE(108,64)-(188,64),10
40 LINE -(188,88),10:LINE-(188,
116)
50 LINE -(104,116)
60 LINE-(96,96)
70 LINE-(70,96)
80 LINE-(74,96)
90 LINE-(74,86)
100 LINE-(114,86)
110 LINE-(132,104)
120 LINE-(140,104)
130 LINE-(140,64),10
140 LINE (76,96)-(76,108)
150 LINE-(100,108)
160 LINE(140,88)-(188,88),10
290 GOTO 290
```

Isso mesmo, ele recria o jipe, agora em outra máquina. Assim como antes, a modalidade escolhida é a 2. A linha 30 desenha uma linha vermelha de (108,64) a (188,64). Como da vez anterior, você não precisa dizer ao computador onde começar a desenhar: ele apenas prossegue de onde estiver no momento. As linhas 40 a 160, portanto, apenas definem os finais de linhas sucessivas.

A linha 150, no entanto, desenha uma linha a partir de um novo ponto inicial; assim, tanto o início como o fim devem ser definidos.

Acrescente agora algumas linhas para o pára-brisa, as rodas e as calotas:

```
180 LINE (140,89)-(188,89)
200 LINE(118,64)-(104,86)
210 CIRCLE(82,116),14,3
230 CIRCLE(82,116),6,2
250 CIRCLE(168,116),14,3
270 CIRCLE(168,116),6,2
```

As linhas 170 e 180 desenham a janela, e a linha 200, o pára-brisa. O comando **CIRCLE** é utilizado pelas linhas 210 e 250 para definir as rodas, e pelas linhas 230 e 270 para desenhar as calotas.

PINTE O SEU JIPE

Para pintar o seu jipe, você não precisa de uma pistola de pintura: a máquina faz isso para você por meio do comando **PAINT**.

O comando **PAINT** preenche com uma cor específica qualquer forma que você tenha desenhado na tela. Ele também possui quatro elementos. Os dois primeiros, como você já deve ter adivinhado, marcam o local onde a pintura deve começar. O terceiro é o código da cor a ser empregada. O quarto é o código de cor da borda, ou da linha, onde a pintura deve acabar.

Acrescente essas linhas ao programa e você verá, na prática, como fica o programa:

```
170 PAINT (142,66),10,10
190 PAINT(77,99),4,4
220 PAINT(82,116),3,3
240 PAINT(82,116),2,2
260 PAINT(168,116),3,3
280 PAINT(168,116),2,2
```

Rode agora o programa e você verá cada seção do jipe sendo preenchida com belas cores.

Não é possível colocar todas as linhas com **PAINT** no final do programa, por uma razão muito importante:

A área que você pretende colorir deve ser fechada completamente por uma série de retas da mesma cor (a cor da borda), ou pela borda da tela.

Qualquer linha de cor diferente que estiver localizada no meio da figura geométrica a ser preenchida deve ser pintada por cima.

As cores disponíveis no **PMODE 3,1** são: verde (1), amarelo (2), azul (3) e vermelho (4). Mas você pode querer pintar o jipe de modo diferente; nesse caso, tudo o que tem a fazer é modificar o número de cor da pintura nas linhas **PAINT** correspondentes.

Outras partes, contudo, podem ser incorporadas ao nosso jipe (uma roda de direção, por exemplo, ou outro equipamento qualquer). Para isso, utilize os comandos **LINE** e **CIRCLE**.

BOMBARDEIOS E EXPLOSÕES

- COMO CRIAR FLASHES NA TELA
- UM PROGRAMA SIMPLES DE BOMBARDEAMENTO AÉREO
- ENRIQUEÇA A AÇÃO COM CHAMAS E EXPLOSÕES

Faça de seus jogos de ação um eletrizante passatempo, aprendendo a obter efeitos gráficos de explosões e incêndios na tela do micro.

É relativamente fácil tornar seus programas de jogos mais espetaculares e emocionantes: para isso, basta acrescentar algumas rotinas com efeitos gráficos especiais. E elas não precisam ser muito complexas para dar a esses jogos um dinamismo bem diferente.

Como você verá nesta lição, existem muitas maneiras de produzir efeitos visuais em BASIC. É preciso apenas saber usar o tipo certo de efeito para o jogo que está sendo programado.

Os efeitos gráficos para chamas e explosões explicados aqui são adequados a todos os tipos de jogos que envolvem a destruição de objetos como construções, carros, aviões, tanques, navios, etc. Alguns deles ficarão ótimos também em jogos espaciais.

Entretanto, como os efeitos de chamas e explosões utilizam o conceito de blocos gráficos, o tamanho máximo do alvo sobre o qual serão colocados é restringido pelas dimensões desses efeitos. Isso significa que você não pode adicioná-los a um programa qualquer; pelo contrário, essa inclusão deve ser cuidadosamente planejada.

Por outro lado, muitos computadores têm comandos embutidos no BASIC que permitem a obtenção de flashes (inversão rápida de cores no vídeo), que são suficientes para muitos tipos de jogos, como os de guerra espacial, e que têm a vantagem de não exigir adaptações para uso em um jogo determinado.

Vamos mostrar aqui como criar, nos micros compatíveis com o Sinclair Spectrum (como o TK-90X), o efeito visual

de um incêndio provocado pela explosão de uma bomba; depois de ir romper fortemente, o fogo se extingue gradualmente, desaparecendo da tela à medida que o prédio desaba. Isso será explicado mais adiante.

Mas, primeiro, precisamos de um avião e de uma bomba. Portanto, digite o programa abaixo:

```
10 FOR N=USR "p" TO USR "q"+7
20 READ a
30 POKE n,a
40 NEXT n
50 DATA 32,16,136,154,155,8,
16,32
60 DATA 0,16,16,120,28,28,0,0
```

Usamos aqui a técnica convencional de blocos gráficos definidos pelo usuário (UDG), disponível no Spectrum. Assim, criamos imagens tanto para o avião quanto para a bomba, armazenadas em uma área especial, definida pelos endereços devolvidos pelas expressões **USR "p"** e **USR "q"** na linha 10. Os códigos correspondentes estão nas declarações **DATA** das linhas 50 e 60, e são colocadas na memória com o **POKE** da linha 30. Ao rodar o programa, nada acontecerá de início. Para verificar se os códigos gráficos em **DATA** foram digitados corretamente, digite a linha de comando abaixo (sem precedê-la com um número de linha).

```
PRINT  AT 10,15;CHR$  159;"   ";
CHR$ 160
```

— que mostrará na tela os dois blocos gráficos definidos.

Em seguida, você precisará de um prédio ou casa para bombardear. Não há necessidade de um novo programa para isto. Apenas modifique as linhas 10 e 50 do programa anterior.

```
10 FOR N=USR "r" TO USR "r"+7
20 READ a
30 POKE n,a
40 NEXT n
50 DATA 255,153,255,153,255,
153,255,255
```

Execute o programa de novo e teste-o, digitando esta linha (novamente, sem colocar número de linha antes):

```
PRINT AT 20,15; CHR$ 161
```

Tendo ficado satisfeito com o resultado visível na tela, digite **NEW** para apagar o programa da memória. Os blocos gráficos definidos antes ficarão armazenados na memória RAM do computador, a menos que você o desligue.

COMECE O BOMBARDEIO

Entre agora o programa do bombardeio por meio das linhas seguintes:

```
10 BORDER 0: PAPER 5: INK 0:
CLS
20 LET a$="                "
200 PRINT  PAPER 4;AT 20,0;a$;
a$;a$;  a$
210 PRINT  INK 1;AT 19,12;CHR$
161;   CHR$ 161;CHR$ 161
```

O que estas linhas fazem, como você verá ao executá-las com o comando **RUN**, é apenas desenhar uma faixa verde, com dezesseis pixels de altura, na parte de baixo da tela; sobre ela ergue-se um armazém ou coisa parecida. A forma como isso é feito ilustra bem o uso de variáveis alfanuméricas no trabalho com gráficos.

Como você precisa de 64 bloquinhos coloridos de verde para fazer o terreno gramado, a linha 20 coloca em a$ uma seqüência de dezesseis espaços em branco. Em seguida, a linha 200 a imprime na tela quatro vezes, começando na linha 20 e prosseguindo na linha 21. Isso economiza a digitação de 64 a$!

A construção é colocada na tela por meio de um quatro **PRINT** do caractere gráfico de código 161 (que é o associado à tecla r, no computador, conforme a definição no programa anterior).

Para fazer o avião voar, só é preciso incluir no programa as seguintes linhas:

```
215 PAUSE 100
220 LET ay=6: LET by=ay
230 FOR x=0 TO 30
240 PRINT AT ay,x;" ";CHR$ 159
250 LET bx=x
260 IF by<19 THEN  PRINT AT by
+1,bx+1;CHR$ 160;AT by,bx;" "
270 LET by=by+1: LET bx=bx+1
280 IF x>29 THEN  PRINT AT ay,
x+1;" "
290 NEXT x
```

Não existe nada de incomum nessa seção do programa: apenas as técnicas convencionais de movimentação de um bloco gráfico na tela, que temos usado em praticamente todos os programas de jogos para o Spectrum. Observe, entretanto, que as linhas 240, 260 e 280 são necessárias para apagar as últimas posições do avião e da bomba, à medida que estes são deslocados pela tela.

A EXPLOSÃO

Vamos agora à parte mais importante do nosso exercício de programação: a explosão. É mais fácil vê-la na tela do que tentar descrevê-la.

Portanto, antes de entrar o restante do programa, digite as linhas abaixo:

```
1000 FOR n=88 TO 80 STEP -1
1010 PRINT AT 10,15;CHR$ 150
1020 POKE 23675,n
1030 PAUSE 50
1040 NEXT n
1050 STOP
```

Mas, espere: não digite o comando **RUN**, ainda. Você poderia destruir a parte do programa que já foi digitada. Ao invés disso, digite **RUN 1000**, para começar a executar só essa parte do programa. Você verá então uma letra G aparecer na tela e depois esvanecer-se gradualmente, até ser substituída pela letra F. Isso acontece porque o laço **FOR...NEXT**, que vai da linha 1000 à 1040, diminui de um em um o valor armazenado na locação de memória 23675, que é o ponto inicial para o seu bloco gráfico UDG. Assim, a letra exibida na tela retrocede gradualmente no alfabeto.

O programa de explosão usa um truque semelhante. Para fins de segurança de seu programa, digite:

```
POKE 23675,88
```

que restaura o valor original do número armazenado na locação de memória

indicada. Em seguida, apague todas as linhas de 1000 a 1050 e digite o restante do programa de bombardeio:

```
 90 POKE 23675,88
 100 FOR n=0 TO 31: READ a:
POKE USR "a"+n,a: NEXT n
 300 FOR e=88 TO 80 STEP -1
 310 POKE 23675,e
 320 PRINT  INK 2;AT 19,12;CHR$
 145 ;CHR$ 147;CHR$ 145: PAUSE
 6
 330 PRINT  INK 2;AT 19,12;CHR$
 147 ;CHR$ 145;CHR$ 147: PAUSE
 6
 340 NEXT e
 400 POKE 23675,88
 500 GOTO 210
8000 DATA 0,0,0,0,0,0,0,0,2,128
,25,126,126,255,255,255
9000 DATA 0,0,0,0,0,0,0,0,4,33,
144,66,231,255,255,255
```

As linhas 100, 8000 e 9000 definem os UDG que você precisa para explosão. A linha 100 lê os dados contidos nas linhas 8000 e 9000, com os códigos numéricos das partes dos gráficos e os coloca na memória através do **POKE.** As linhas 300 e 340 usam a técnica descrita anteriormente para retirar uma imagem e substituí-la por outra.

Existe, entretanto, uma diferença muito importante. Os UDG que criam a explosão são baseados nos caracteres gráficos B e D — respectivamente, **CHR$ 145** e **147.** À medida que eles forem desaparecendo da tela, você não vai querer que eles sejam substituídos por outras letras do alfabeto. Assim, os primeiros oito elementos de cada declaração **DATA,** representando os caracteres A e C — **CHR$ 144** e **146,** respectivamente — são todos zeros.

Deste modo, ao invés de obter A e C substituindo B e D na tela, você obtém espaços em branco. Se você quiser ver como isto funciona em "câmara lenta", inclua a linha abaixo (não esqueça de retirá-la, depois):

```
335 PAUSE 100
```

A linha 400 é necessária, evidentemente, para restaurar o valor original da memória 23675, que era 88, antes que o programa se repita de novo.

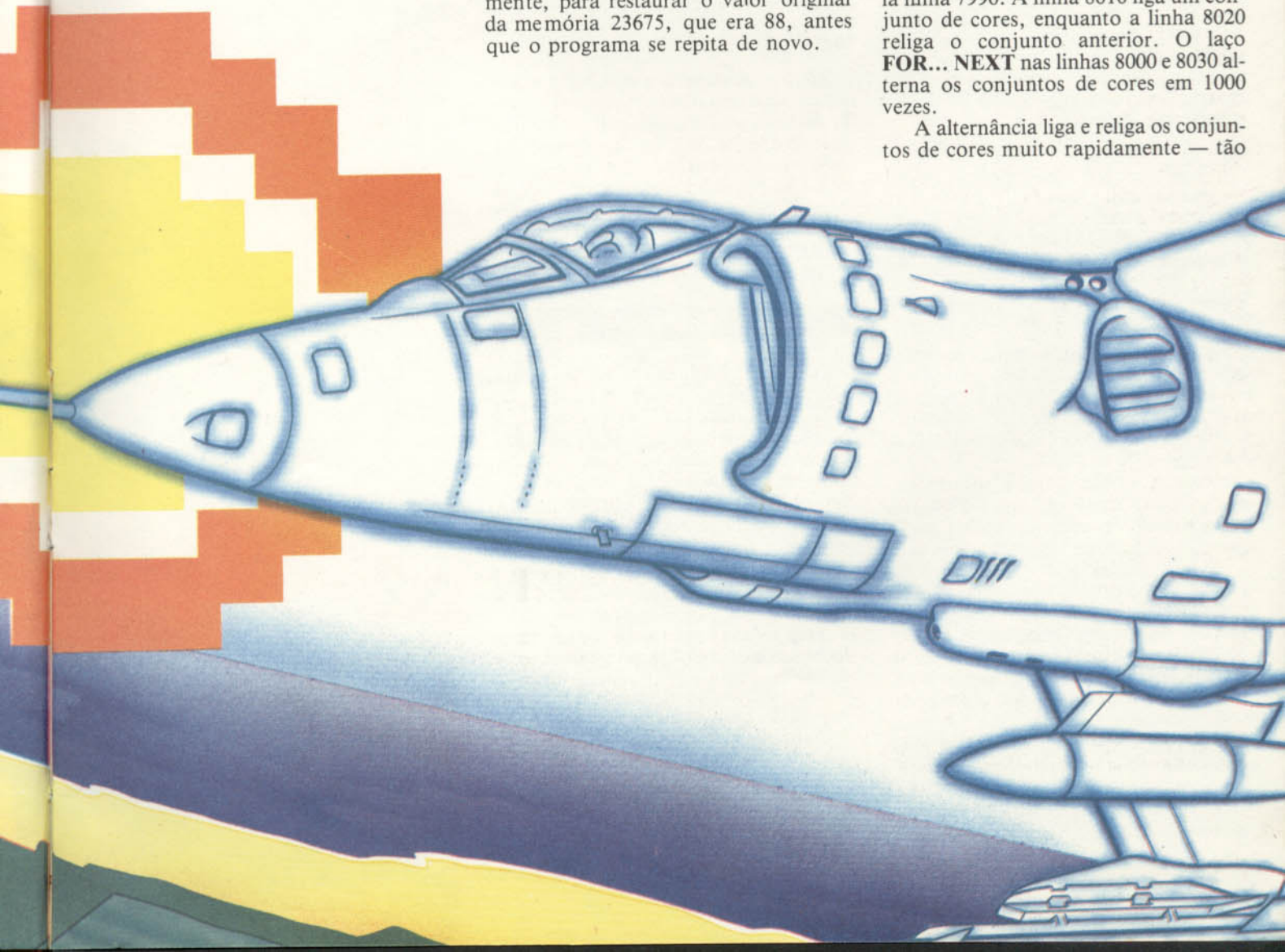
T

Os computadores compatíveis com essa linha têm excelentes recursos para a obtenção de efeitos visuais dramáticos, usando programas bastante curtos. Tente digitar e executar o programa abaixo:

```
7980 PMODE 3,1
7990 PCLS
8000 FOR F=1 TO 1000
8010 SCREEN 1,0
8020 SCREEN 1,1
8030 NEXT F
8040 CLS
```

Observe que o programa coloca faixas coloridas que migram através da tela; esse efeito é causado pela alternância super-rápida dos diferentes conjuntos de cores no computador. Ele pode ser usado, entre outras coisas, para assimilar o fim de uma fase do jogo, como uma blindagem que foi penetrada.

O modo gráfico é definido pela linha 7980, e a seguir a tela gráfica é limpa pela linha 7990. A linha 8010 liga um conjunto de cores, enquanto a linha 8020 religa o conjunto anterior. O laço **FOR... NEXT** nas linhas 8000 e 8030 alterna os conjuntos de cores em 1000 vezes.

A alternância liga e religa os conjuntos de cores muito rapidamente — tão

rapidamente, na verdade, que o aparelho de TV nunca tem tempo de mudar as cores na tela completamente antes de entrar em ação o outro conjunto. Como conseqüência, apenas uma faixa estreita da tela é mudada para cada cor.

Se você pretende utilizar esse programa na forma de uma sub-rotina dentro de outro jogo (no qual funcionará apenas com um dos comandos **PMODE** descritos na página 144) ou com o programa de batalha espacial apresentado a seguir, terá que fazer algumas alterações. Primeiro, apague as linhas 7980 e 7990; em seguida, adicione a linha:

```
8500 RETURN
```

Veja agora o programa a seguir:

```
10 PMODE 1,1
20 DIM A(3),B(3),M(3)
200 FOR K=1536 TO 2016 STEP 32
210 READ A,B: POKE K,A:POKE K+1
,B
220 NEXT
230 GET(0,0)-(15,15),A,G
240 GET(0,16)-(15,31),M,G
250 PCLS
260 MX=120:MY=191:PX=120:PY=20
270 PUT (PX,PY)-(PX+15,PY+15),A
,PSET
280 SCREEN 1,0
290 PUT (MX,MY)-(MX+15,MY+15),B
,PSET
300 MY=MY-4
310 IF MY<36 THEN GOSUB 8000:GO
TO 260
320 PUT (MX,MY)-(MX+15,MY+15),M
,PSET
330 GOTO 290
9000 DATA 252,63,3,192,15,240,6
1,124,58,172,245,95,213,87,213,
87
9010 DATA 0,128,0,128,0,128,2,1
60,10,168,0,192,3,240,15,60
```

O programa mostra uma nave extraterrestre na iminência de ser atingida por um míssil. No momento que este alcança o alvo, entra em ação a sub-rotina definida anteriormente, que move faixas coloridas pela tela.

Quando usado com a sub-rotina, ele faz faixas com qualquer **PMODE** que se queira, de modo que pode ser adicionado ao final de um programa, sem que seja necessário alterar as linhas 8000 a 8040.

Eis aqui uma adaptação do programa para fazer faixas que resulta em um efeito visual mais elaborado, embora os gráficos na tela não permaneçam intocados como anteriormente. Você pode digitá-lo e executá-lo do jeito que está, ou incorporá-lo a um outro programa — como o da animação da nave espacial — na forma de sub-rotina. Neste caso, você precisará alterar o programa, apagando a linha 7990 e adicionando ao final uma linha com comando **RETURN** como foi feito anteriormente.

```
7990 PMODE 3,1
8000 FOR F=1 TO 3
8010 FOR K=0 TO 1
8020 SCREEN 1,K
8030 FOR J=1 TO 4
8040 PCLS J
8050 NEXT J
8060 NEXT K
8070 NEXT F
8080 CLS
```

Sob a forma de sub-rotina, ele funcionará em qualquer um dos modos gráficos com conjuntos de duas cores **(PMODE 0, 2 e 4)** e de quatro cores **(PMODE 1 e 3)**. Além de mudar os conjuntos de cores, a rotina também limpa a tela.

As linhas 8030 a 8050 são um laço **FOR... NEXT** que colore a tela com as cores disponíveis no conjunto ativo.

Uma última variação do programa:

```
7990 PMODE 3,1
8000 FOR F=1 TO 5
8010 SCREEN 1,0
8020 FOR K=1 TO 200: NEXT K
8030 SCREEN 1,1
8040 FOR K=1 TO 200: NEXT K
8050 NEXT F
8060 CLS
```

Você pode escolher entre digitá-lo assim como está (como um programa autônomo) ou na forma de sub-rotina (digitando a linha 7990 e adicionando um comando **RETURN** ao seu final).

A tela piscará rapidamente, pois as linhas 8020 e 8040 inserem pequenos retardos de tempo, dando oportunidade suficiente para a mudança de cor entre um conjunto e outro.

CHAMAS

Podemos obter efeitos visuais ainda melhores através da programação de gráficos animados. Eis aqui uma idéia para fazer a animação gráfica das chamas de um incêndio, que você poderia, por exemplo, superpor a um alvo atingido por uma bomba. O programa foi escrito para o **PMODE 1**, e não pode ser usado, do jeito que está, com jogos em outros **PMODE**. Digite-o e execute-o:

```
10 PMODE 1,1
20 DIM B(3),E1(3),E2(3)
30 FOR K=1536 TO 2016 STEP 32
40 READ A,B:POKE K,A:POKE K+1,B
50 NEXT
60 GET(0,0)-(15,15),E1,G
70 GET(0,16)-(15,31),E2,G
80 SCREEN 1,0
250 PCLS:HX=124:HY=146
8000 FOR N=0 TO 15
8010 PUT (HX,HY+N)-(HX+15,HY+15
),E1,PSET
8020 FOR K=1 TO 100:NEXT
8030 PUT (HX,HY+N)-(HX+15,HY+15
),E2,PSET
8040 FOR K=1 TO 100:NEXT
8050 PUT (HX,HY+N)-(HX+15,HY+15
),B,PSET
8060 NEXT
9000 DATA 0,12,192,0,3,195,63,2
52,63,252,255,255,255,255,255,2
55
9010 DATA 0,48,12,3,195,0,48,12
,252,63,255,255,255,255,255,255
```

Dois conjuntos de chamas são colocados na tela pelo comando **PUT**.

Os dados para os conjuntos de chamas estão nas declarações **DATA** das linhas 9000 e 9010. Eles são colocados com comandos **POKE** na tela, com as linhas 30 e 50. O programa está escrito em uma modalidade gráfica de quatro cores. As linhas 60 e 70 capturam, através do comando **GET**, as formas gráficas na tela e as armazenam nos conjuntos E1 e E2, dimensionados na linha 20.

As linhas 8000 e 8060 animam as chamas. Quando o programa passa pelo laço **FOR...NEXT**, os conjuntos E1 e E2 surgem na tela, e apagam quando se coloca sobre eles o conjunto de espaços

em branco, B, com **PUT**. O topo das chamas é abaixado um pouco de cada vez, pelo + **N** nas linhas com comando **PUT**.

Você pode usar esse programa como sub-rotina do programa abaixo, que simula um bombardeio, mas não se esqueça de alterar a linha 20 para como ela aparece no programa: edite-a com o comando **EDIT** ou digite-a novamente. Apague a linha 80, digitando o número 80, seguido de <**ENTER**>.

```
20 DIM A(3),B(3),H(3),E1(3),E2(
3)
200 FOR K=1536 TO 2016 STEP 32
210 READ A,B:POKE K,A:POKE K+1,
B
220 NEXT
230 GET(0,0)-(15,15),A,G
240 GET(0,16)-(15,31),H,G
250 PCLS:LINE(0,163)-(255,191),
PSET,BF
260 HX=124:HY=146:PX=0:PY=40:B=
0
```

```
270 PUT (HX,HY)-(HX+15,HY+15),H
,PSET
280 SCREEN 1,0
290 PUT (PX,PY)-(PX+15,PY+15),B
,PSET
300 PX=PX+4
310 PUT(PX,PY)-(PX+15,PY+15),A,
PSET
320 IF PX=20 THEN B=1:BX=PX+8:B
Y=PY+8
330 IF B=1 THEN PRESET(BX,BY):P
RESET(BX+2,BY):BX=BX+2:BY=BY+2:
PSET(BX,BY,4):PSET(BX+2,BY,4)
340 IF BY=148 THEN GOSUB 8000:B
Y=0:GOTO 250
350 GOTO 290
8070 RETURN
9020 DATA 0,0,2,0,130,128,162,1
60,170,170,162,160,130,128,2,0
9030 DATA 0,3,12,51,60,243,255,
255,255,255,85,85,86,149,86,149
```

Eis aqui duas versões que permitirão usar a sub-rotina em jogos programados em outras modalidades gráficas (**PMODE**). A primeira é um programa que deve ser usado com as **PMODE 3** e **4**; mude apenas o número apropriado.

```
10 PMODE 3,1
20 DIM B(6),E1(6),E2(6)
30 FOR K=1536 TO 2496 STEP 64
40 READ A,B:POKE K,A:POKE K+1,B
45 POKE K+32,A:POKE K+33,B
50 NEXT
60 GET (0,0)-(15,15),E1,G
70 GET (0,16)-(15,31),E2,G
80 SCREEN 1,0
250 PCLS:HX=124:HY=146
8000 FOR N=0 TO 15
8010 PUT(HX,HY+N)-(HX+15,HY+15)
,E1,PSET
8020 FOR K=1 TO 100: NEXT
8030 PUT (HX,HY+N)-(HX+15,HY+15
),E2,PSET
8040 FOR K=1 TO 100:NEXT
8050 PUT(HX,HY+N)-(HX+15,HY+15)
,B,PSET
8060 NEXT
9000 DATA 0,12,192,0,3,195,63,2
52,63,252,255,255,255,255,255,2
55
9010 DATA 0,48,12,3,195,0,48,12
,252,63,255,255,255,255,255,255
```

Em segundo lugar, temos um programa adequado para uso com **PMODE 2**:

```
10 PMODE 2,1
20 DIM B(6),E1(6),E2(6)
30 FOR K=1536 TO 2496 STEP 32
40 READ A: POKE K,A: POKE K+16,
A
50 NEXT
60 GET(0,0)-(15,15),E1,G
70 GET(0,16)-(15,31),E2,G
80 SCREEN 1,0
250 PCLS:HX=124:HY=146
8000 FOR N=0 TO 15
8010 PUT(HX,HY+N)-(HX+15,HY+15)
,E1,PSET
8020 FOR K=1 TO 100:NEXT
8030 PUT (HX,HY+N)-(HX+15,HY+15
),E2,PSET
8040 FOR K=1 TO 100: NEXT
8050 PUT(HX,HY+N)-(HX+15,HY+15)
,B,PSET
8060 NEXT
9000 DATA 2,128,25,126,126,255,
255,255
9010 DATA 4,33,144,66,231,255,2
55,255
9020 DATA 0,12,192,0,3,195,63,2
52,63,252,255,255,255,255,255,2
55
```

Os computadores compatíveis com a linha MSX têm excelentes recursos (tais como os *sprites*), para se conseguir efeitos visuais dramáticos, usando-se programas bastante curtos. Tente digitar e executar o programa abaixo:

```
7990 SCREEN 0:KEY OFF
8000 FOR F=1 TO 300
8010 COLOR 10,10,10
8020 COLOR 4,4,4
8030 NEXT F
8040 COLOR 15,4,4
```

O modo de texto da tela é definido pela linha 7990 (**SCREEN 0**). A linha 8010 estabelece uma cor de fundo (10) para a tela, com a mesma cor de moldura e de frente (por isso todos os números são iguais); a linha 8020 especifica um segundo conjunto de cores para a tela. O laço **FOR...NEXT** nas linhas 8000 e 8030 alterna os conjuntos de cores trezentas vezes. A linha 8040 volta tudo à situação original de cor da tela, antes de interromper o programa (se você interrompê-lo usando as teclas <**CTRL**> <**STOP**>, antes de terminar o laço de repetição, terá que digitar pelo teclado, ou com a tecla F1, o comando existente na linha 8040).

Como a alternância das cores é muito rápida, apenas uma faixa estreita da tela é mudada para cada cor. Esse efeito só pode ser conseguido, no MSX, com a tela de texto. Podemos introduzi-lo como fase final de um programa de animação gráfica. Acrescente as seguintes linhas ao programa:

```
10 SCREEN 2,2
200 FOR K=1 TO 24
210 READ A:READ B
220 A$=A$+CHR$(A):B$=B$+CHR$(B)
230 NEXT
240 SPRITE$(2)=A$:SPRITE$(1)=B$
250 MX=120:MY=191
260 PX=120:PY=20
270 PUT SPRITE 2,(PX,PY)
280 PUT SPRITE 1,(MX,MY)
290 PUT SPRITE 1,(MX,MY)
300 MY=MY-4
310 IF MY<36 THEN GOTO 7990
320 PUT SPRITE 1,(MX,MY)
330 GOTO 290
```

```
8050 RUN
9000 DATA 252,0,3,0,15,0,61,2,5
8,10,24,5,0,213,3,213,15
9010 DATA 0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0
9020 DATA 63,128,192,128,240,12
8,124,160,172,168,95,192,87,240
,87,60
```

Os dois objetos a serem movimentados na tela (um míssil e uma nave) são definidos pelos códigos gráficos das linhas 9000 a 9020, montados nos sprites 1 e 2. As linhas 270 e 290 colocam esses sprites na tela, animando o sprite 1 (o míssil) através do ciclo repetido entre 290 e 330.

Quando o míssil atinge a nave (linha 310), o programa de efeito visual de faixas começando em 7990 é acionado. A linha 8050 manda executar o programa principal desde o começo, em virtude da necessidade de redefinição da tela gráfica na linha 10.

Em seguida, temos uma adaptação do programa para fazer faixas. Você pode digitá-lo e executá-lo como está, ou adicioná-lo a outro programa, como o da animação da nave espacial.

```
7990 SCREEN 0:KEY OFF
8000 FOR F=1 TO 15
8005 FOR K=1 TO 15
8010 COLOR F,K,K
8020 COLOR K,F,F
8025 NEXT K
8030 NEXT F
8040 COLOR 15,4,4
8050 RUN
```

As linhas 8000 a 8030 formam dois laços **FOR...NEXT** que colorem a tela com cores disponíveis na tela do texto MSX. A última variação do programa é:

```
7990 SCREEN 0:KEY OFF
8000 FOR F=1 TO 5
8010 COLOR 4,10,10
8015 FOR K=1 TO 150:NEXT K
8020 COLOR 10,4,4
8025 FOR K=1 TO 150:NEXT K
8030 NEXT F
8040 COLOR 15,4,4
8050 RUN
```

A tela pisca rapidamente, em cores porque as linhas 8015 e 8025 inserem pequenos retardos de tempo, dando oportunidade para o vídeo mudar de cor entre um conjunto e outro.

FOGO!

Efeitos visuais ainda melhores podem ser obtidos através da programação de sprites nos gráficos animados. Eis aqui uma idéia para fazer a animação gráfica de um incêndio.

O programa foi escrito para a tela gráfica **SCREEN 2**. Esse comando também é usado para definir o tamanho do sprite. Digite o comando **NEW** e entre as seguintes linhas:

```
10 SCREEN 2,2
30 FOR K=1 TO 32
40 READ A:READ B
45 A$=A$+CHR$(A):B$=B$+CHR$(B)
50 NEXT
60 SPRITE$(2)=A$
70 SPRITE$(1)=B$
75 SPRITE$(0)=STRING$(32,255)
250 HX=124:HY=146
8000 PUT SPRITE 0,(HX,HY+16),12
8005 FOR N=0 TO 15
8010 PUT SPRITE 2,(HX,HY+N),6
8020 FOR K=1 TO 100:NEXT
8025 PUT SPRITE 2,(HX,HY+16)
8030 PUT SPRITE 1,(HX,HY+N),6
8040 FOR K=1 TO 100:NEXT
8060 NEXT
9000 DATA 0,0,192,12,3,195,63,4
8,63,252,255,255,255,255,255,25
5
9010 DATA 255,255,255,255,255,2
55,255,255,255,255,255,255,255,
255,255,255
9020 DATA 12,48,0,3,195,0,252,1
2,252,63,255,255,255,255,255,25
5
9030 DATA 255,255,255,255,255,2
55,255,255,255,255,255,255,255,
255,255,255
```

Dois conjuntos de chamas, armazenados em sprites de tamanho 2 (ou seja, 16 por 16 pixels), são colocados na tela pelo comando **PUT**. Os dados para os conjuntos de chamas estão nas declarações **DATA** das linhas 9000 a 9030. Eles são lidos pelas declarações **DATA** na linha 40 e colocados nas variáveis **A$** e **B$**, na linha 45. Após isso, os sprites números 1 e 2 recebem o conteúdo desses cordões. Um terceiro sprite (um bloco gráfico uniformemente cheio) também é definido na linha 75. A cor que esse bloco de sprite vai assumir ao ser colocado na tela é determinada pelo comando **PUT** na linha 8000 (no caso, azul, que é o código 4).

As linhas 8000 a 8060 animam as chamas. Cada vez que o programa passa pelo laço **FOR...NEXT**, os dois sprites 1 e 2 são colocados na tela. O topo das chamas é abaixado um pouco de cada vez, pelo **+N** nas linhas com o comando **PUT**, de modo que dá a impressão de diminuição do fogo.

Esse programa pode ser usado como uma sub-rotina do programa a seguir, que simula um bombardeio: um prédio é alvejado. Adicione as linhas:

```
200 FOR K=1 TO 32
210 READ A:READ B
215 D$=D$+CHR$(A):E$=E$+CHR$(B)
220 NEXT
230 SPRITE$(3)=D$
240 SPRITE$(4)=E$
250 LINE(0,163)-(255,191),12,BF
260 HX=124:HY=146:PX=0:PY=40:B=
0
270 PUT SPRITE 4,(HX,HY),9
300 PX=PX+4
310 PUT SPRITE 3,(PX,PY),14
320 IF PX=20 THEN B=1:BX=PX+8:B
Y=PY+8
330 IF B=1 THEN PRESET (BX,BY):
PRESET(BX+2,BY):BX=BX+2:BY=BY+2
:PSET(BX,BY),1:PSET(BX+2,BY),1
340 IF BY=148 THEN PUT SPRITE 4
,(HX,HY+16):PRESET (BX,BY):PRES
ET (BX+2,BY):GOSUB 8000:BY=0:GO
TO 250
350 GOTO 300
8070 RETURN
9040 DATA 0,0,0,0,2,12,2,12,130
,60,130,60,162,255,162,255
9050 DATA 170,255,170,255,162,8
5,162,85,130,86,130,86,2,86,2,8
6
9060 DATA 0,3,0,3,0,51,0,51,128
,243,128,243,160,255,160,255
9070 DATA 170,255,170,255,160,8
5,160,85,128,149,128,149,0,149,
0,149
```

Nesse trecho suplementar, mais dois sprites, 3 e 4, são definidos pelas linhas 200 a 240, a partir dos códigos gráficos armazenados nas declarações **DATA** nas linhas 9040 a 9070.

A linha 250 traça um retângulo de cor verde (o chão gramado); a linha 270 coloca dentro dele o desenho da casa. A animação gráfica do avião e da bomba caindo é realizada pelas linhas 310 e 350. Na linha 340 é chamada a sub-rotina definida a partir de 8000, que recebe um **RETURN** na linha 8070.

Não é tão fácil criar efeitos visuais no Apple se o compararmos com outros micros. Ao invés de representar os desenhos através de números binários convertidos para decimais e colocados na memória de vídeo ("1" representaria ponto aceso, e "O", ponto apagado), temos que criar uma tabela que pode ter até 256 figuras, e uma vez que esteja num lugar apropriado da memória pode-se desenhar e mover as figuras com o comando **DRAW**, sendo a cor definida por **HCOLOR**. O programa a seguir move uma chama na base da tela.

```
10   HOME : HGR : SCALE= 1: ROT=
 64
20   POKE 232,0: POKE 233,3
30   FOR K = 768 TO 868
40   READ A: POKE K,A
50   NEXT
250 HX = 124:HY = 146
8000  FOR N = 1 TO 15
8010  HCOLOR= 5
8020  FOR K = 1 TO 3
8030  DRAW 1 AT HX,HY + N: HCOL
OR= 0
8040  DRAW 1 AT HX,HY + N: HCOL
OR= 5
8050  DRAW 1 AT HX + 3,HY + N +
 2: HCOLOR= 0
8060  DRAW 1 AT HX + 3,HY + N +
 2: HCOLOR= 5
8070  NEXT K
8080  NEXT N
8090  TEXT
9000  DATA  3,0,8,0,64,0,0,1,15
0,18,54,37,36,36,4,11,54,54,54,
46,36
9010  DATA  36,11,54,54,46,36,4
,100,100,11,54,54,54,37,36,4,10
0,100,11,54
9020  DATA  54,54,46,36,100,100
,100,11,22,54,54,37,36,100,100,
11,54,54,54,37
9030  DATA  36,4,0,45,56,255,21
9,35,109,73,45,37,63,63,63,63,3
9,45,45,45
9040  DATA  45,45,46,44,46,45,3
7,63,231,219,63,255,219,39,64,7
3,44,46,5,0
```

O mesmo incêndio é desenhado e apagado rapidamente em duas posições diferentes, dando a impressão de fogo. As chamas desaparecem lentamente na parte inferior do vídeo. O programa funciona da seguinte maneira:

A tabela de figuras está nas linhas **DATA** (9000 a 9040). Ela é colocada na memória pela repetição do comando **POKE** da linha 40, devida ao laço **FOR...NEXT** das linhas 30 e 50. A linha 20 informa onde está a tabela de figuras na memória. A linha 10 limpa a tela, estabelece o modo gráfico de alta resolução, fixa o tamanho (**SCALE**) e a orientação (**ROT**) da figura.

As linhas 8000 a 8060 animam as chamas e as fazem desaparecer. Cada vez que o programa repete as linhas que ficam entre **FOR** e **NEXT**, as chamas são desenhadas e apagadas em duas posições diferentes da tela. Elas vão baixando lentamente devido ao **+N** nas linhas **HPLOT**, de forma que o fogo parece acabar. Isso acontece porque o desenho invade o espaço reservado para texto na parte inferior da tela, onde podemos colocar dados gráficos sem que eles apareçam. Para entender melhor, experimente introduzir a linha:

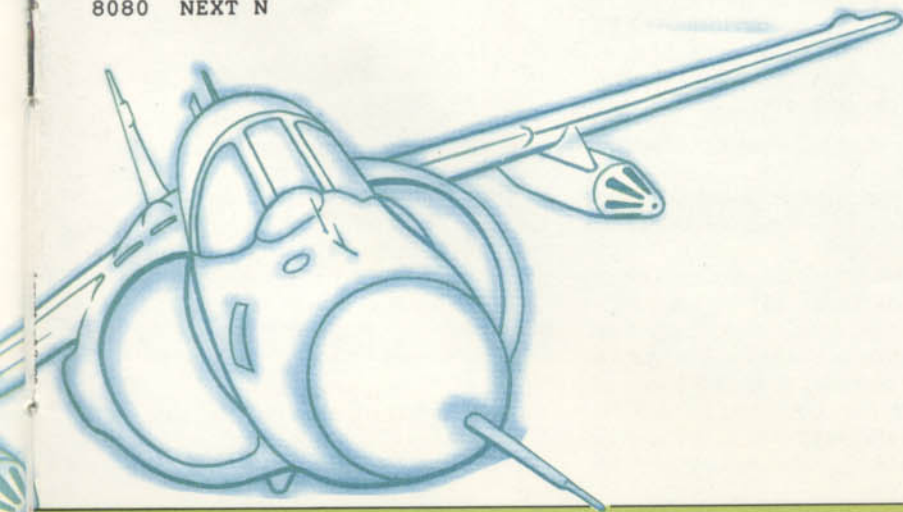
```
15 POKE -16302,0
```

Essa linha elimina o espaço reservado para textos da parte inferior da tela, tornando visíveis quaisquer gráficos ali colocados. Rodando o programa com essa modificação, você verá as chamas ocuparem posições cada vez mais baixas sem desaparecer. Não se esqueça de apagar essa linha digitando 15 e **<RETURN>**. Nosso incêndio ficará mais interessante se for produzido por uma sub-rotina dentro de um jogo, como no exemplo a seguir. Após acrescentar essas linhas, você verá uma pequena casa ser bombardeada e pegar fogo. Não se esqueça de apagar a linha 15.

```
55 I = 0
60  FOR J = 8674 TO 15810
    STEP 10 24
70  READ A(I),B(I)
80 I = I + 1
100  NEXT
190 I = 0
200  FOR J = 8674 TO 15810
     STEP 1 024
210  POKE J,B(I): POKE J +
     1,A(I)
215 I = I + 1
220  NEXT
235  FOR K = 1 TO 1000: NEXT
250 HX = 139:HY = 153:PX = 259:
     PY = 40:B = 0
290 PX = PX - 4
300  HCOLOR= 6: DRAW 2 AT PX,PY
310  HCOLOR= 0: DRAW 2 AT PX,PY
320  IF PX = 239 THEN B =
     1:BX = PX:BY = PY + 8
330  IF B = 1 THEN  HCOLOR= 0:
     HPLOT BX,BY: HPLOT BX -
     2,BY:BX = BX - 2:BY =
     BY + 2: HCOLOR= 3: HPLOT
     BX,BY: HPLOT BX - 2 ,BY
340  IF BY = 152 THEN  GOSUB
     8000 :BY = 0: GOTO 190
350  GOTO 290
8090  RETURN
9050  DATA   0,3,12,51,60,243,
     255,255,255,255,85,85,86,
     149,86,149
```

A casa é desenhada pelos comandos **POKE** da linha 210. O padrão do desenho é obtido da linha 9050 e guardado nas variáveis indexadas **A** e **B**. Programar desenhos dessa maneira é mais fácil que definir tabelas, mas quando se deseja movimento e velocidade (como nas chamas e no avião) é preciso usar as tabelas e o comando **DRAW**.

O formato do avião já estava na tabela do programa anterior. As chamas são a figura 1 da tabela, e o avião é a figura 2. Essas figuras são desenhadas respectivamente pelos comandos **DRAW 1** e **DRAW 2**.

Para parar o programa aperte simultaneamente **<CTPL>** e C. Mesmo que não apareça nada na tela, digite **TEXT** e aperte **<RETURN>** e tudo voltará ao normal.

OS COMANDOS READ E DATA

Se você pretende evitar a digitação de programas longos e cansativos, convém preparar seu computador para a leitura de dados armazenados internamente em declarações DATA.

As variáveis são o grande responsável pela versatilidade do computador. Atribuir valores a elas é, a maior parte das vezes, simples e direto: basta, por exemplo, dizer **LET X = 5.** Mas há ocasiões em que a quantidade de informações que se deseja utilizar em um programa é muito grande. Nesses momentos, é preciso recorrer às declarações **DATA**, e aos comandos **READ** e **RESTORE** (esses comandos não estão disponíveis no BASIC de alguns micros).

A palavra **DATA** (dados, em inglês) é utilizada para muitas coisas. Assim, tudo o que é fornecido (''dado'' pelo programador ou pelo usuário) ao computador é incluído nessa categoria. Nesse artigo, porém, vamos usar a expressão dados apenas em seu sentido mais restrito, ou seja, informações incluídas em um programa por meio da declaração **DATA.**

A declaração **INPUT** serve para atribuir valores a uma variável. Mas ela é útil somente enquanto a informação estiver sendo fornecida ao computador cada vez que o programa for rodado.

Existem valores fixos, porém, que não precisam ser entrados ou alterados pelo operador, e podem, assim, ser incorporados permanentemente ao programa. Neste caso, entra em cena o segundo método de atribuição: a declaração **LET.** Mas, quando as informações são em grande quantidade, o emprego de comandos **LET** torna-se cansativo.

Veja a seguir um exemplo onde uma lista fixa de cabeçalhos é utilizada para imprimir um documento:

```
10 LET A$="DIA"
20 LET B$="SEMANA"
30 LET C$="MES"
40 LET C$="ANO"
50 PRINT A$,B$,C$,D$
```

Veja como **DATA** pode ser utilizada com o mesmo objetivo e com uma redução no número de linhas necessárias:

```
10 READ A$,B$,C$,D$
20 PRINT A$,B$,C$,D$
100 DATA DIA,SEMANA,MES,ANO
```

Em programas curtos não há quase diferença entre usar **LET** ou **DATA.** Agora, se você quiser trabalhar com cinqüenta cabeçalhos, em vez de quatro, o primeiro programa precisará de 46 declarações **LET** a mais, enquanto o segundo exigirá apenas uma ou duas linhas a mais.

Nos compatíveis com o Spectrum (como o TK-90X), as palavras **DATA** devem estar sempre entre aspas. Nos outros modelos as aspas são dispensáveis. A linha 100 passaria a ser escrita assim:

```
100 DATA  "DIA","SEMANA","MES",
    "ANO"
```

COMO FUNCIONAM READ E DATA

Quando o computador se depara com uma instrução **READ**, ele passa a procurar em todo o programa onde se encontra a primeira declaração **DATA**. Então ele atribui o primeiro item de **DATA** à variável correspondente no comando **READ**. No programa acima, o cordão "DIA", na declaração **DATA**, é atribuído à variável **A$**, "SEMANA" é atribuído a **B$**, etc. O computador ignora qualquer declaração **DATA** a menos que uma declaração **READ** faça com que ele a leia.

```
10 DATA ALEMANHA
20 READ A$,B$
30 DATA BRASIL,ITALIA,ESPANHA
40 READ C$,D$
```

É mais fácil ler o programa quando todos os **DATA** estão reunidos em um mesmo lugar. E existe uma regra inabalável: os comandos **DATA** devem aparecer na ordem em que o computador vai lê-los, por meio da instrução **READ.**

DIFERENTES TIPOS DE DATA

Além de cordões alfanuméricos, as declarações **DATA** podem conter números. A linha Sinclair Spectrum também

- COMO FUNCIONAM AS DECLARAÇÕES **READ** E **DATA**
- DIFERENTES TIPOS DE **DATA**
- UTILIZE **DATA** PARA FAZER UMA LISTA TELEFÔNICA
- A DECLARAÇÃO **RESTORE**
- APLICAÇÕES PARA LISTAS DE **DATA**
- COMO DESENHAR GRÁFICOS SIMPLES A PARTIR DE **DATA**

permite que você utilize variáveis e funções dentro de declarações **DATA**.

```
DATA "RECIFE",256,a*5,SQR(num)
```

No comando **READ** as variáveis devem ser colocadas na mesma ordem que os itens relativos a **DATA**. A linha **DATA** acima pode ser lida com o comando:

```
READ A$,N,X,Y
```

A primeira variável é uma variável *alfanumérica,* para coincidir com o primeiro item de **DATA.** Mas os três itens seguintes são variáveis *numéricas.*

É fácil cometer erros quando se usa **DATA** em um programa. Os principais problemas ocorrem quando não se possui um número suficiente de itens ou quando se tenta ler com **READ** uma variável de outro tipo, em **DATA.** Se a linha **DATA** acima tivesse sido digitada incorretamente, tal como **DATA 256, RECIFE, a★5,** então **A$** receberia o valor 256! Embora errada, ela não seria rejeitada (a não ser no Spectrum) pois o computador interpreta o valor numérico 256, neste caso, como o cordão alfanumérico "256".

Em contrapartida, se o computador tentasse ler o cordão "RECIFE" e o atribuísse à variável númerica N, poderia perceber o erro e responder com uma mensagem tal como: *"tipo inadequado"* (erro TM, ou TYPE MISMATCH), ou *"dado incorreto"* (erro BAD DATA, ou INVALID DATA, dependendo do computador); ou poderia supor que "RECIFE" é um nome de variável não definido anteriormente; sua resposta, neste caso, seria uma mensagem de erro como: *"variável não encontrada".*

Outro erro comum é colocar itens a menos em **DATA.**

UMA LISTA TELEFÔNICA

Eis aqui um programa que utiliza um laço para a leitura interna de dados, através das declarações **READ** e **DATA.** É um programa muito simples para elaborar uma lista telefônica particular.

O programa abaixo roda apenas no TRS-Color. Para utilizá-lo nos micros TRS-80, modifique os números do **PRINT** nas linhas 50 e 60, para 640.

```
5 CLS
10 PRINT @70,"DIRETORIO TELEFON
ICO"
12 PRINT
20 INPUT "DIGITE O NOME";R$
30 FOR J=1 TO 5
40 READ N$,T$
50 IF N$=R$ THEN PRINT @320,"O
NUMERO DE ";R$;" E: ";T$:END
60 IF N$="FIM" THEN PRINT @320,
R$;" NAO ESTA NA LISTA"
70 NEXT J
500 DATA ALCINO,384-4276,JUNIOR
,997-4036,RODRIGO,52-5114,GERTR
UDES,5666-99994,FIM,FIM
```

```
 5 CLS
10 PRINT AT 2,6;"GUIA TELEFON
ICO"
 15 RESTORE
 20 INPUT "Digite o nome",R$
 30 FOR J=1 TO 5
 40 READ N$,T$
 50 IF N$=R$ THEN  PRINT AT 10
,1;"O numero de ";N$;" e ";T$
: GOTO 80
 60 IF N$="FIM" THEN  PRINT AT
10,3;R$;" NAO ESTA NA LISTA"
 70 NEXT J
 80 PRINT AT 12,0;"Quer outro
numero (S/N)?"
 90 PAUSE 0
100 IF INKEY$="S" THEN  GOTO 5
110 IF INKEY$="N" THEN  GOTO
2000
500 DATA "ZULEICA","22-6400","
MARIA", "33-7237","MARCELO","3
4-4009","RICARDO","13-71824","
FIM","FIM"
```

```
10 LOCATE 12,3:PRINT"Lista tele
fônica"
12 PRINT:PRINT
15 RESTORE
20 INPUT"Digite o nome";R$
30 FORJ=1TO5
40 READ N$,T$
50 IFN$=R$ THEN LOCATE 5,15:PRI
NT"O número de ";N$;" é ";T$:GO
TO 80
60 IFN$="END" THEN LOCATE 5,15:
PRINTR$;" não está na lista!"
70 NEXTJ
500 DATA ESTELA,321054,JOSÉ,529
884,CELI,326747,FERNANDO,511934
,END,END
```

```
5  HOME
10  VTAB 3: HTAB 12: PRINT "Lis
ta telefonica"
12  PRINT : PRINT
20  INPUT "Digite o nome: ";R$
30  FOR J = 1 TO 5
40  READ N$,T$
50  IF N$ = R$ THEN  VTAB 15: H
TAB 5: PRINT "O numero de ";N$;
" e ";T$: END
60  IF N$ = "END" THEN  VTAB 15
: HTAB 5: PRINT R$;" nao esta n
a lista!"
70  NEXT J
500  DATA  FERNANDO, 510536, MA
RILIA, 543286,JOAQUIM,28999,ROB
ERTA,719852,END,END
```

Você pode utilizar os nomes e os números dos telefones dos seus amigos na linha **DATA** e colocar tantos itens **DATA** quanto desejar; mas não se esqueça de ajustar o contador de laços na linha 30.

A lista de itens em **DATA** é finalizada com as palavras FIM, FIM, para a linha 60 checar se já foi alcançado o fim da lista. Se o programa tiver lido a lista em **DATA** até o fim, isso significa que o nome não foi encontrado, e o programa imprimirá uma mensagem para dizer isto a você. FIM (ou "FIM" no Spectrum) deve ser entrado duas vezes porque a linha 40 lê dois itens **DATA** de cada vez.

Os cordões em **DATA** podem conter espaços, mas não vírgulas. Se você precisar entrar um cordão **DATA** com uma vírgula, coloque o cordão entre aspas:

```
10 READ N$,A$;B$,C$
20 DATA JOSE DOS CAMPOS,
   "AV. BRASIL,23",
   CAMPINAS,13100
```

A primeira linha do endereço deve ser escrita entre aspas por causa da vírgula depois de AV. BRASIL.

COMO USAR A DECLARAÇÃO RESTORE

Com os métodos vistos até agora, um programa poderá ler uma lista de **DATA** apenas uma vez, a menos que você o rode de novo. Para ler mais uma vez o conjunto de itens em **DATA** você deve incorporar ao seu programa a declaração **RESTORE** (*restaurar*, em inglês). Se você quiser consultar os números dos telefones dos amigos sem rodar o programa, substitua **END** ou **STOP** na linha 50 por **GOTO 80** e acrescente:

O programa listado abaixo roda apenas no TRS-Color. Para utilizá-lo em micros da linha TRS-80, modifique o número do **PRINT** na linha 80 para 832.

```
80 PRINT @416,"VOCE QUER OUTRO
NUMERO(S/N)?"
90 K$=INKEY$:IF K$="" THEN GOTO
 90
100 IF K$="S" THEN GOTO 5
110 END
```

```
 80 PRINT AT 12,0;"Quer outro
 numero (S/N)?"
 90 PAUSE 0
100 IF INKEY$="S" THEN  GOTO 5
110 IF INKEY$="N" THEN  GOTO
2000
```

```
80 LOCATE 3,21:PRINT"Você quer
outro número? (S/N)"
90 K$=INKEY$:IF K$="" THEN 90
100 IF K$="S" THEN 5
110 END
```

```
80  VTAB 22: HTAB 1: PRINT "Voc
e deseja mais algum numero? (S/
N)";
90  GET K$
100  IF K$ = "S" THEN 5
110  END
```

Mas o que acontecerá quando você rodar o programa? Na primeira vez, ele funcionará bem. Mas, se você pressionar uma tecla para outra repetição da consulta, ele falhará devido à falta de mais itens em **DATA.** Mas existe uma solução para o problema. Acrescente:

```
15 RESTORE
```

Desta vez o programa continuará funcionando toda vez que se repetir, pois o comando **RESTORE** instrui o computador a voltar ao início da lista em **DATA.** É uma boa idéia colocar sempre uma declaração **RESTORE** no início de um programa que você estiver desenvolvendo. Isso faz com que você não fique sem nenhuma **DATA,** ao tentar testá-lo. Se, mais tarde, você quiser remover a linha **RESTORE,** bastará colocar uma declaração **REM** após a linha em que ela está, para lembrá-lo.

Com o comando **RESTORE** você pode reutilizar uma lista **DATA** sempre que necessário. Ele é particularmente útil quando se quer ler várias vezes uma mesma lista de dados para pesquisar um determinado item, como no caso do programa para a lista telefônica.

APLICAÇÕES PARA DATA

As declarações **DATA** são úteis para todos os tipos de programa e você certamente já as utilizou em desenhos de labirintos e outros efeitos.

Freqüentemente, um grande número de linhas **DATA** é utilizado para conter todo o texto necessário a um jogo de aventuras, ou em programas aplicativos simples. Pode-se também armazenar questionários com todas as perguntas e respostas em linhas **DATA.** E os jogos de fliperama em BASIC as utilizam para definir tanto os caracteres como os planos de fundo.

Programadores experientes empregam **DATA** para linguagem Assembly ou em programas em código de máquina (veja a primeira lição de *Código de Máquina*). Tais programas consistem de um laço **FOR...NEXT** curto que lê os códigos de máquina em uma lista **DATA** e usa o comando **POKE** para inseri-los na memória.

Qualquer programa que contenha textos padronizados, figuras ou funções pode fazer um largo uso das listas **DATA.** Empregadas de maneira cuidadosa, essas listas constituem um poderoso instrumento de programação.

UTILIZE **DATA** PARA GRÁFICOS

As declarações **DATA** servem igualmente para programas de gráficos, para definir coordenadas, desenhar ou evocar caracteres gráficos predefinidos na memória ROM. Sua utilidade, porém, diminui quando o que se quer é traçar formas regulares onde as coordenadas ou o formato dos gráficos podem ser calculados. Um exemplo disso pode ser encontrado no programa para o TRS-Color, onde os padrões repetitivos no topo das muralhas de um forte são calculados.

As declarações **DATA** no programa abaixo foram deliberadamente divididas em várias linhas. Assim, se você quiser examinar ou modificar o programa futuramente e achar os itens em **DATA** que correspondem às variáveis do programa, poderá seguir uma a uma as instruções **READ;** mas para um programa longo isto será muito cansativo. O melhor será dividir as declarações **DATA** em grupos, e — se você quiser evitar a memorização de tudo o que está lá dentro — utilizar comandos **REM** para explicar para que serve cada grupo de declarações **DATA.**

O programa seguinte desenha uma casa:

```
10 CLS:COLOR 4,5
20 FOR A=0 TO 8
30 FOR B=1 TO 17
40 READ C
50 VPOKE (171+A*40)+B,C
60 NEXT B
70 NEXT A
500 DATA 32,32,32,32,32,32,32,3
2,32,32,32,32,219,219,219,32,32
510 DATA 32,32,32,32,32,32,32,3
2,32,32,32,219,219,219,219,219,
32
520 DATA 32,32,32,32,32,32,32,3
2,32,32,219,219,219,219,219,219
,219
530 DATA 32,32,32,199,219,219,3
2,32,32,32,32,222,219,219,219,2
19,219
540 DATA 32,32,199,219,219,219,
219,32,221,32,32,219,219,219,21
9,219,32
550 DATA 32,32,215,215,215,215,
215,219,219,193,32,32,222,215,2
19,32,32
560 DATA 32,32,215,16,215,202,2
15,215,21,215,32,32,32,215,32,3
2,32
570 DATA 32,32,215,215,215,202,
215,215,215,215,32,32,32,215,32
,32,32
580 DATA 195,195,195,195,195,19
5,195,195,195,195,195,195,195,1
95,195,195,195
```

Os números em **DATA** fornecem as coordenadas das várias partes da casa (linhas 500 a 580). A parte principal do programa estabelece os laços **FOR...NEXT** para a leitura (comando **READ** na linha 40) das partes pertinentes em **DATA** e as desenha, colocando os códigos de caracteres na página de memória da tela, por meio do comando **VPOKE.**

Este programa utiliza **READ...DATA** para desenhar uma ponte. Se você entrar o programa e rodá-lo em estágios, será bem mais fácil ver o que está acontecendo, e assim verificar se o digitou corretamente:

```
10 FOR t=74 TO 80 STEP 3
20 PLOT 35,t
30 DRAW 175,0,-2.5
40 NEXT t
```

Este segmento do programa utiliza um laço **FOR...NEXT** para ajudar a traçar três pontos próximos ao lado esquerdo da tela; em seguida, ele desenha uma linha em forma de arco a partir de cada ponto. A linha 30 significa: "desenhe até um ponto localizado 175 pixels à direita do pixel inicial". O parâmetro -2,5 dá à linha sua forma curva, evitando a linha reta.

```
 100 FOR n=18 TO 39
 110 READ a
 120 PLOT n,45
 130 DRAW 0,a
 132 PLOT n+188,45
 134 DRAW 0,a
 140 NEXT n
1000 DATA 70,70,67,67,70,70,60,
60,57,57,60,60,57,57,60,60,70,7
0,67,67,70,70
```

Esta é a seção que desenha as torres. O laço entre as linhas 100 e 140, mais o número 45 (pixels da parte inferior da tela), nas linhas 120 e 132, traçam os pontos na parte inferior de cada um dos conjuntos de linhas verticais que definem a torre.

Então, as linhas 110 e 1000 assumem o comando. A linha 110 diz ao computador para ler, na linha 1000, a altura (novamente em pixels) de cada uma das 22 linhas verticais. Assim, a primeira linha será 0,70 — isto é, vertical e com 70 pixels de altura; a segunda linha será 0,70, a terceira 0,67 e assim por diante. Se você quiser ver o que está acontecendo, tente inserir uma linha temporária tal como **135 PAUSE 100** após a linha 134.

```
300 PLOT 0,75
310 DRAW 255,0-0.1
320 PLOT 0,78
330 DRAW 255,0,-0.1
```

Essas linhas desenham a pista de rolamento e -0.1 produz uma ligeira curva para cima.

```
 400 FOR r=62 TO 182 STEP 20
 410 PLOT r,78
 420 READ b
 430 DRAW 0,b
 440 NEXT r
1010 DATA 42,55,63,65,63,55,42
```

Essas, por sua vez, produzem os cabos verticais entre o arco e a pista de rolamento; o laço **FOR...NEXT** ajuda a traçar as posições iniciais dos cabos, enquanto as linhas 420 e 1010 regulam as suas respectivas alturas.

Se você quiser que o programa rode de novo, automaticamente, acrescente essas linhas:

```
  5 CLS:RESTORE
450 GOTO 5
```

A linha 5 limpa a tela e permite que o programa leia novamente o **DATA.**

TRABALHE COM INDICADORES RESTORE

Até agora, o único problema com **DATA** se resumiu a que a informação precisou sempre ser chamada na mesma seqüência, começando do mesmo lugar. Utilizando o **RESTORE**, é possível voltar ao início de uma lista, mesmo que não se tenha atingido o fim. Mas como se salta para o meio de uma lista?

Nos computadores das linhas Sinclair Spectrum e MSX, existe um modo para se responder a essa questão. Em vez de uma lista apenas, você pode utilizar várias, direcionando assim o computador para a lista adequada. Tente acrescentar essas linhas extras ao programa para o Spectrum:

```
    6 INPUT "PONTE ANTIGA OU MOD
    ERNA?";   y$
    7 IF y$="a" THEN  RESTORE
    1000
    8 IF y$="m" THEN  RESTORE
    2000
    9 IF y$<>"a" AND y$<>"m"
    THEN  GOTO   5
  10 FOR t=74 TO 80 STEP 3
  20 PLOT 35,t
  30 DRAW 175,0,-2.5
  40 NEXT t
2000 DATA 83,84,85,86,87,88,89,
90,90,90,90,90,90,90,90,89,88,8
7,86,85,84,83
2010 DATA 42,55,63,65,63,55,42
```

Os números que seguem o comando **RESTORE** são conhecidos como indicadores de **RESTORE.** Eles direcionam o computador para uma lista **DATA.** No programa acima, se você pressionar a tecla O, o indicador **RESTORE** será definido com o 1000, na linha 7.

Se você quiser que o micro vá para a primeira lista **DATA** disponível no programa, não será necessário um indicador: usar o **RESTORE** sem um número de linha é o mesmo que restaurar o número da primeira linha **DATA.**

Indicadores de restauração são muito úteis em programação de jogos. Numa aterrissagem, por exemplo, você poderia escrever uma rotina para checar se houve uma colisão ou um pouso seguro. As listas **DATA** poderiam então conter informações para gerar sons para as duas alternativas, e o indicador **RESTORE** serviria para indicar a lista correta.

Outra vantagem do **RESTORE** indexado para definir os elementos de uma figura é que fica bem mais fácil acrescentar novos elementos a um desenho terminado; se quisermos adicionar um muro à casinha da ilustração abaixo, bastará colocar mais linhas **DATA:**

```
80 LOCATE 0,21:PRINT"Pressione
qualquer tecla para uma casa co
m muro!";
90 IFINKEY$<>""THENRESTORE520:G
OTO10:ELSEGOTO90
590 DATA 203,203,203,203,203,20
3,203,203,203,203,203,203,203,2
03,203,203,203
600 DATA 203,203,203,203,203,20
3,203,203,203,203,203,203,203,2
03,203,203,203
```

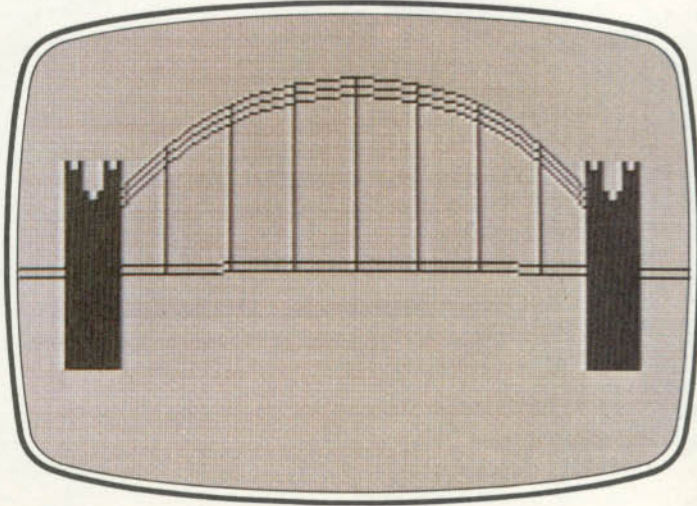
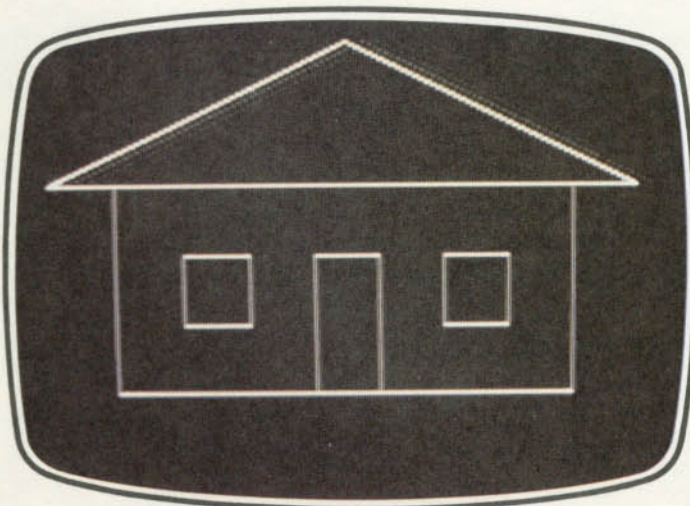
Feito o desenho, o computador perguntará se você quer uma casa com muro. Se você pressionar qualquer tecla, o comando **RESTORE 520** na linha 90 indicará, para a próxima execução do programa (a partir da linha 10), que as linhas **DATA** lidas pelos comandos **READ** na linha 40 deverão começar na linha 520, e não em 500, que é o seu início físico. De acordo com uma nova seqüência de caracteres gráficos (que termina com as linhas adicionais 590 e 600), o desenho será diferente do anterior.

O programa a seguir construirá um castelo:

```
10 PCLEAR 4
20 PMODE 4,1
30 PCLS 5
40 SCREEN 1,1
50 READ SX,SY
60 LINE -(SX,SY),PSET
70 FOR K=1 TO 18
80 READ X,Y
90 LINE -(X,Y),PRESET
100 NEXT K
270 GOTO 270
500 DATA 64,160
510 DATA 64,60,32,60,48,40,64,6
0,32,60,32,160,110,160,110,120
520 DATA 152,120,152,160,228,16
0,228,60,212,40,196,60,228,60
530 DATA 196,60,196,160,196,74
```

As linhas de 10 a 40 deixam a tela para gráficos de alta resolução pronta para desenhar. A linha 270 a mantém ligada. As linhas 50 e 60 são um tanto incomuns. As coordenadas do ponto inicial do castelo são lidas, e uma linha em branco é desenhada em fundo branco para o ponto inicial. Algumas vezes, essa capacidade para desenhar linhas "invisíveis" torna-se bastante útil, pois elas podem se juntar às linhas visíveis em um esquema contínuo de programação.

Da linha 70 à 100, é desenhado o contorno do castelo. No final desta parte do programa foram lidos 38 itens de **DATA.** Agora acrescente essas linhas e construa algumas muralhas:

```
110 FOR K=1 TO 33
120 LET X=X-4
130 LINE-(X,Y),PRESET
140 IF Y=74 THEN LET Y=78 ELSE
```

```
LET Y=74
150 LINE -(X,Y),PRESET
160 NEXT K
```

Provavelmente você está tentando adivinhar o que aconteceu à linha **READ.** Esta é uma ocasião em que as instruções **DATA** e **READ** não representam economia de trabalho, pois você precisaria de 66 itens de **DATA** para definir os cantos da muralha. Programando desta forma, você não precisará digitar todas as linhas **DATA** extras.

Essas linhas desenharão as janelas:

```
170 FOR K=1 TO 8
180 READ X,Y
190 LINE (X,Y) - (X+4,Y),PRESET
200 LINE (X+2,Y-2)-(X+2,Y+6),PR
ESET
210 NEXT K
540 DATA 46,80,46,120,210,80,21
0,120,86,90,170,90,80,132,178,1
32
```

Você não precisa do **DATA** para finalizar as janelas. Como elas são todas do mesmo tamanho, apenas um conjunto de **DATA** é necessário. Acrescente agora essas linhas, rode o programa e veja o que acontece:

```
220 FOR K=1 TO 4000
230 NEXT K
240 CLS: PRINT @33,"PRESSIONE Q
UALQUER TECLA PARA   CONTINUAR"
250 LET IN$=INKEY$:IF IN$="" TH
EN GOTO 250
270 GOTO 30
```

Quando você pressionar uma tecla qualquer, obterá OD ou a mensagem de erro "out of data". A razão disso é que o programa chegou ao final da lista **DATA** — não existe mais nenhuma depois dele. Mas você não precisa digitar todos os **DATA** novamente. Acrescente só esta linha, que utiliza a declaração **RESTORE** para instruir o computador a retornar ao início da lista **DATA:**

```
260 RESTORE
```

A casa da página anterior foi desenhada por um Apple II a partir de quatro linhas **DATA**. Para fazer a ponte, o Spectrum utilizou **PLOT** e **DRAW**, junto com **DATA**. Ao lado, castelo construído em etapas pelos TRS-Color.

Desenhar uma casinha nos micros da linha Apple é facílimo quando utilizamos **DATA.** Digite o programa abaixo:

```
10 HGR
20 HCOLOR=3
30 FOR I=1 TO 4
40 READ XC,YC,LX,LY
50 HPLOT XC,YC TO XC+LX,YC TO
   XC+LX,YC+LY TO XC,YC+LY TO
   XC,YC
60 NEXT I
80 FOR I=1 TO 3
90 READ XO,YO,XD,YD
100 HPLOT XO,YO TO XD,YD
110 NEXT I
400 DATA 20,50,160,80
410 DATA 90,80,20,50
420 DATA 40,80,30,30
430 DATA 130,80,30,30
440 DATA 0,50,200,50
450 DATA 200,50,100,0
460 DATA 100,0,0,50
```

Os elementos em **DATA** fornecem as coordenadas para as diferentes partes da casa.

A parte principal do programa liga o modo de alta resolução gráfica (HGR) e lê os dados contidos em **DATA** por intermédio do laço **FOR...NEXT** nas linhas 30 a 60 e 80 a 110.

Os dados contidos nas linhas **DATA** que vão de 400 a 430 contêm quatro números cada e servem para traçar retângulos de qualquer tamanho e em qualquer posição na tela (rotina **HPLOT** na linha 50). Essa estratégia é vantajosa, pois você pode notar na ilustração que a casa tem no mínimo quatro retângulos diferentes (a casa, a porta e as duas janelas). Assim, os dois primeiros números em **DATA** são as coordenadas XC e YC do ponto de origem do retângulo na tela (seu canto superior esquerdo). Os dois números seguintes são: o comprimento do lado paralelo ao eixo horizontal (LX) e o do lado paralelo ao eixo vertical (LY). Assim, o laço **FOR...NEXT** das linhas 30 e 60 lê sucessivamente quatro conjuntos de dados e traça os quatro retângulos.

Já o traçado do triângulo exige uma técnica diferente. O laço que vai das linhas 80 a 110 lê 4 valores: XO, YO (coordenadas de origem de uma reta) e XD, YD (coordenadas do seu ponto final), e traça essa reta, na linha 90.

Uma vez que você tenha rodado o programa, talvez queira repeti-lo mais uma vez. Para isso, tente acrescentar as linhas seguintes, rode o programa e veja o que acontece:

```
120 HOME
130 PRINT "QUER VER DE NOVO
    (S/N)?":GET R$
140 IF R$="N" THEN TEXT:END
150 GOTO 10
```

Como vimos antes, se você pressionar uma tecla qualquer obterá OD ou a mensagem de erro "out of data". A razão disso é que o programa chegou ao final da lista **DATA.** Felizmente, você não precisa digitar todos os **DATA** novamente. Apenas acrescente a linha seguinte, que utiliza a declaração **RESTORE** para instruir o computador a retornar ao início da lista **DATA:**

```
150 RESTORE:GOTO 10
```

ORGANIZE MELHOR OS COMANDOS DATA EM UM PROGRAMA

As linhas **DATA** têm quase sempre uma coisa em comum: qualquer que seja a informação que elas transportam, leva-se muito tempo para digitá-las e fazê-las funcionar. Tudo correrá bem na primeira vez, mas pode ser que você precise retornar posteriormente a um programa e não consiga entender mais nada do que foi colocado lá.

Organizar sistematicamente as linhas **DATA** do programa toma apenas um pouco mais de tempo, mas pode poupar séculos no resultado final.

Onde o **DATA** definir um gráfico por blocos ou algo parecido, organize as linhas do programa para corresponder diretamente às linhas do gráfico. Se o **DATA** segue um formato repetitivo (tal como os dados em uma lista telefônica, por exemplo), classifique cada entrada exatamente do mesmo modo nas linhas do programa.

PONHA ORDEM EM SUAS CONTAS

Assim como os grandes computadores comerciais, seu micro pode ser usado para fazer cálculos e manter registros financeiros. Veja como fazer isso e facilite sua vida.

Neste artigo, examinaremos um programa de contabilidade doméstica, que foi planejado para fornecer respostas a perguntas como: onde foi parar o dinheiro do salário? São apresentadas versões para todos os computadores, com exceção dos compatíveis com o ZX-81.

Para atualizar a sua contabilidade, você deve "alimentá-la" uma vez por mês com seus rendimentos e despesas.

O programa cria uma espécie de diário contendo uma coluna para os rendimentos e sete colunas para as despesas sob diferentes rubricas. As subdivisões referentes às despesas podem ser modificadas, caso necessário; para isso, altere os dizeres (identificação das categorias de despesa) na declaração **DATA** pertinente, quando digitá-las, junto com o programa. Mas existem algumas limitações: a coluna de rendimentos deve aparecer sempre em último lugar; ao mesmo tempo, é necessário que haja um total de oito colunas.

O programa deve ser digitado em duas seções: a primeira, com o próprio programa; e a segunda, com todas as informações com as quais você o alimentou até o momento da sua entrada. Isto significa que você precisará de dois nomes para os programas.

Para gravar corretamente o programa em fita, siga o procedimento normal de sua máquina (comando **SAVE** ou **CSAVE**). Para recarregar o programa, siga novamente o procedimento usual para carregar jogos gravados em fitas.

Quando você rodar o programa, o menu principal lhe dará sete opções:

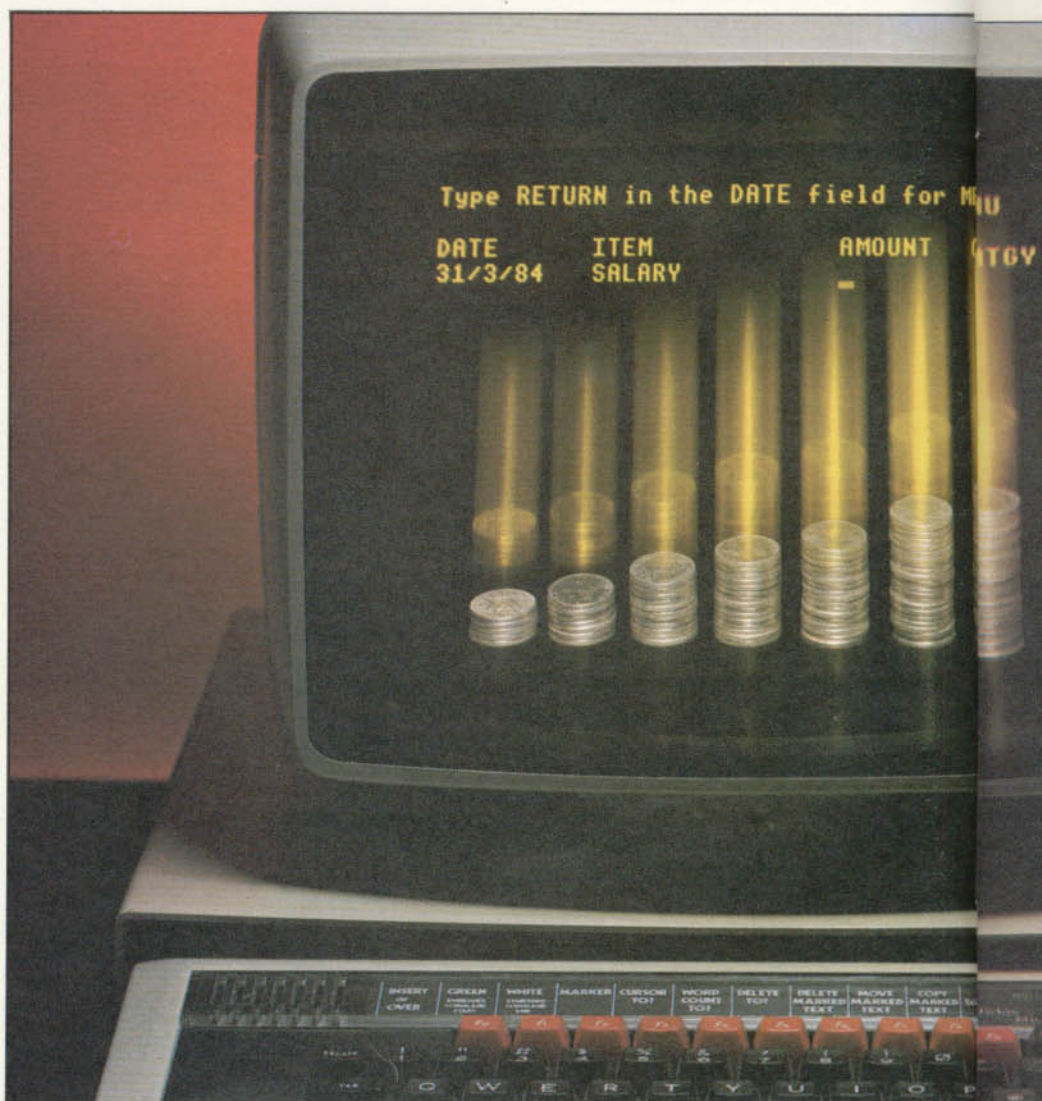
1 - Entrar dados
2 - Ver dados
3 - Gravar em fita
4 - Carregar da fita
5 - Imprimir
6 - Alterar um dado
7 - Encerrar o programa

Para fazer um lançamento, pressione a tecla 1 quando o menu principal aparecer. Não acione ainda **<ENTER>** ou **<RETURN>**. A cada lançamento, o computador pedirá detalhes como data, descrição, valor e categoria.

Digite a informação na ordem dada, pressionando **<ENTER>** ou **<RETURN>** após cada dado do lançamento. Espere que o computador pergunte

por um novo dado e pressione **<ENTER>** ou **<RETURN>**, trazendo de volta o menu.

Para examinar um lançamento, pressione a tecla 2 quando o menu principal aparecer. Não tecle **<ENTER>** ou **<RETURN>**. O computador exibirá um índice mostrando as várias categorias. Para selecionar uma delas digite o número apropriado (não tecle **<ENTER>** ou **<RETURN>**, e o micro listará os lançamentos que existem para aquela categoria.

Na versão para o Spectrum, a tela mostrará a questão "Continua?" se houver espaço insuficiente para exibir todos os lançamentos de uma vez. Neste caso, não pressione o N nesse ponto: prossiga até o final da listagem.

Quando terminar, acione **<ENTER>** ou **<RETURN>** e volte ao menu principal.

COMO MODIFICAR UM LANÇAMENTO

Quando você teclar o 6 para a opção de alterar um lançamento, o computa-

- DIGITE E ARMAZENE O PROGRAMA
- AS OPÇÕES DO MENU
- COMO ENTRAR LANÇAMENTOS NOS REGISTROS
- ATUALIZE OS REGISTROS
- VERIFICAÇÃO DO BALANÇO
- IMPRESSÃO DOS RESULTADOS
- ARMAZENE SUAS FINANÇAS EM FITA

dor mostrará uma lista com todos os lançamentos já efetuados.

Você pode se movimentar para trás e para a frente dentro da lista, utilizando os sinais de prontidão que aparecerão na tela. O computador também dirá como editar (modificar) um lançamento. Uma vez que você tenha digitado **<ENTER>** ou **<RETURN>**, após fazer a alteração, o computador trará de volta o menu principal. Para uma segunda alteração, você deve selecionar a opção 6 novamente. O comando para a opção de impressão funciona de forma tão simples quanto um interruptor de luz — um liga/desliga acionado quando se tecla a opção 5 (também sem **<ENTER>** ou **<RETURN>**) do menu principal. O computador pedirá que você tecle S, se quiser usar a impressora, ou N se não quiser. Se você pressionar a tecla S, o menu principal voltará automaticamente à tela e, a partir desse momento, tudo o que normalmente apareceria na tela, usando-se a opção 2, passará a ser mandado para a impressora. Se quiser interromper essa opção, pressione a tecla 5 novamente e responda N.

Se você não tiver uma impressora conectada ao computador, tome cuidado para não pressionar a tecla S. O Spectrum irá ignorar as instruções neste caso, mas em qualquer um dos outros três computadores você poderá perder todas as informações já inseridas.

T

```
10 PMODE 0:PCLEAR 1:CLEAR 10000
20 DIM TE$(200),AM(200),DA$(200
),CT$(8),CA(200)
30 FOR N=1 TO 8:READ CT$(N):NEX
T
40 DATA MANUTENCAO DA CASA,LAZE
R ,ALUGUEL,VESTUARIO,AUTOMOVEL,
FERIAS,OUTROS,RENDA
50 U1$="#########,":U2$="######
##,"
60 CLS4: PRINT @8,"MENU PRINCIP
AL";:PRINT @70,"1- ENTRAR DADOS
    ";:PRINT @134,"2- VER ENTRA
DAS    ";:PRINT @198,"3- GRAVAR
 EM FITA  ";
70 PRINT @262,"4- CARREGAR
    ";:PRINT @326,"5- IMPRIMIR
       ";:PRINT @390,"6- MUDAR E
NTRADA   ";:PRINT @454,"7- SAID
A          ";
80 A$=INKEY$:IF A$<"1" OR A$>"7
" THEN 80
90 ON VAL(A$) GOSUB 1000,2000,3
000,4000,5000,6000,7000
100 GOTO 60
1000 CLS: IF NU>200 THEN PRINT
@264,"MEMORIA LOTADA !":PLAY "T
10ABCDEFGP1P1":RETURN
1010 GOSUB 1160
1020 GOSUB 1250:INPUT"DATA ";DA
$(NU)
1030 IF DA$(NU)="" THEN RETURN
1040 IF LEN(DA$(NU))>8 THEN 102
0
1050 PRINT @L,DA$(NU);
1060 GOSUB 1250:LINEINPUT"ITEM
?";TE$(NU):IF LEN(TE$(NU))>25 T
HEN 1060
1070 A$=LEFT$(TE$(NU),9):PRINT@
L+14-LEN(A$)/2,A$;
1080 GOSUB 1250:INPUT"QUANTIA "
;A
1090 IF A>9999999 OR A<0 THEN 1
080
1100 PRINT @L+19,USING U2$;A;:A
M(NU)=A
1110 GOSUB 1250:INPUT"CATEGORIA
 ";CA$
1120 GOSUB 1180:IF F=0 THEN 111
0
1130 CA(NU)=NM:PRINT @L+29, LEF
T$(CT$(CA(NU)),3);
1140 IF NM<>8 THEN GT=GT+A
1150 NU=NU+1: L=L+32:IF L=448 T
HEN 1000 ELSE 1020
1160 L=64: PRINT @2,"DATA" TAB(
10)"ITEM" TAB(19) "QUANTIA" TAB
(29)"CAT";
1170 RETURN
1180 IF VAL(CA$)<>0 THEN 1230
1190 F=0:FOR N=1 TO 8
1200 IF CA$=LEFT$(CT$(N),LEN(CA
$)) THEN F=F+1:NM=N
1210 NEXT: IF F>1 THEN F=0
1220 RETURN
1230 IF VAL(CA$)>8 THEN F=0:RET
URN
1240 NM=VAL(CA$):F=1:RETURN
1250 PRINT @448," ":PRINT @449,
;:RETURN
2000 CLS3:FOR N=1 TO 8
2010 PRINT @69+N*32,N;MID$("- "
+CT$(N)+STRING$(12," "),1,20);
2020 NEXT
2030 TT=0: PRINT @449,"QUE CATE
GORIA ?";
2040 A$=INKEY$:IF A$<"1" OR A$>
"8" THEN 2040
2050 NM=VAL(A$)
2060 IF PT=1 THEN PRINT#-2,CHR$
(13):PRINT#-2,TAB(21-LEN(CT$(NM
))/2);CT$(NM):PRINT#-2,"  DATA"
TAB(20)"ITEM"TAB(39)"QUANTIA"
2070 GOSUB 2280
2080 FOR NN=0 TO NU
2090 IF CA(NN)<>NM THEN 2150
2100 IF PT=1 THEN PRINT#-2,USIN
GF$;DA$(NN);TE$(NN);AM(NN)
2110 PRINT @L,DA$(NN);:A$=LEFT$
(TE$(NN),13):PRINT @L+16-LEN(A$
)/2,A$;:PRINT @L+23,USING U2$;A
M(NN);:TT=TT+AM(NN)
2120 L=L+32:IF (L=448 AND NM<>8
) OR (L=352 AND NM=8) THEN PRIN
T @465,"CONT ?";ELSE GOTO 2150
2130 A$=INKEY$:IF A$="" THEN 21
30
2140 GOSUB 2280
```

```
2150 NEXT: IF PT=1 THEN PRINT#-
2,CHR$(13): IF NM<>8 THEN PRINT
#-2,TAB(28);:PRINT#-2,USING"TOT
AL ="+U1$;TT
2160 PRINT @463,USING "TOTAL ="
+U1$;TT;
2170 IF NM<>8 THEN 2250
2180 IF PT=0 THEN 2220
2190 PRINT#-2,TAB(21);:PRINT#-2
,USING"RENDA TOTAL ="+U1$;TT
2200 PRINT #-2,TAB(16);:PRINT#-
2,USING"DESPESA TOTAL ="+U1$;GT
:PRINT#-2,TAB(35)"           "
2210 PRINT#-2,TAB(26);:PRINT#-2
,USING"BALANCO ="+U1$;TT;
2220 PRINT @392,USING "RENDA TO
TAL ="+U1$;TT;
2230 PRINT @419,USING "DESPESA
TOTAL ="+U1$;GT;
2240 PRINT @461,USING"BALANCO =
"+U1$;TT-GT ;
2250 A$=INKEY$:IF A$="" THEN 22
50
2260 IF A$<>CHR$(13) THEN 2000
2270 RETURN
2280 L=64:CLS NM:PRINT @(33-LEN
(CT$(NM)))/2,CT$(NM);
2290 PRINT @34,"DATA";:PRINT @4
5,"ITEM";:PRINT @56,"QUANTIA";
2300 FOR N=32 TO 416 STEP 32
2310 POKE N+1032,122+NM*16:POKE
 N+1046,117+16*NM
2320 NEXT:RETURN
3000 CLS:MOTORON:PRINT @64,"POS
ICIONE O GRAVADOR E PRESSIONE<E
NTER>"
3010 A$=INKEY$:IF A$="" THEN 30
10
3020 MOTOROFF:PRINT @65,"PRESSI
ONE 'REC' NO GRAVADOR     E TEC
LE <ENTER>"
3030 A$=INKEY$:IF A$="" THEN 30
30
3040 CLS: PRINT @65,;:INPUT"NOM
E DO ARQUIVO ";DA$
3050 OPEN "O",#-1,DA$
3060 PRINT #-1,NU
3070 FOR N=0 TO NU-1
3080 PRINT #-1,DA$(N),TE$(N),AM
(N),CA(N)
3090 NEXT: CLOSE#-1:RETURN
4000 CLS: PRINT @65,;:INPUT"NOM
E DO ARQUIVO";DA$
4010 MOTORON:PRINT @64,"POSICIO
NE O GRAVADOR E TECLE     <ENTER
>"
4020 A$=INKEY$:IF A$="" THEN 40
20
4030 MOTOROFF:GT=0:OPEN "I",#-1
,DA$
4040 PRINT @129,"ACHEI ";DA$
4050 INPUT #-1,NU
4060 FOR N=0 TO NU-1
4070 INPUT #-1,DA$(N),TE$(N),AM
(N),CA(N)
4080 IF CA(N)<>8 THEN GT=GT+AM(
N)
4090 NEXT: CLOSE #-1: RETURN
5000 CLS: PRINT @65, "VOCE QUER
 IMPRIMIR ? (S/N)";
5010 A$=INKEY$:IF A$<>"S" AND A
$<>"N" THEN 5010
5020 PRINT "OK":IF A$="N" THEN
PT=0:RETURN
5030 F$="%              %%
              %    "+U2$:PT=1:RETU
RN
6000 IF NU=0 THEN RETURN
6010 CLS: PRINT"  DATA"TAB(11)"
ITEM"TAB(19)"QUANTIA"TAB(29)"CA
T"
6020 PRINT @417,"PRESSIONE <";C
HR$(94);"> PARA PROSSEGUIR
OU <";CHR$(95);"> PARA RETORNAR
OU"
6030 PRINT @481,"A BARRA DE ESP
ACOS PARA EDITAR";:M=0:GOTO 608
0
6035 KK$=INKEY$:IFKK$=""THEN603
5
6040 IF KK$=CHR$(8) AND M>0 THE
N M=M-1:GOTO 6080
6050 IF KK$=CHR$(94) AND M<NU-1
 THEN M=M+1:GOTO 6080
6060 IF KK$=CHR$(32) THEN 6100
6070 GOTO 6035
6080 PRINT@64,USING"%          % %
       %#########, %  %";DA$(M);
LEFT$(TE$(M),8);AM(M);LEFT$(CT$
(CA(M)),3)
6090 GOTO 6035
6100 IF CA(M)<>8 THEN GT=GT-AM(
M)
6110 INPUT"NOVA DATA ";D$:IF D$
="" THEN 6130
6120 IF LEN(D$)>8 THEN 6110 ELS
E DA$(M)=D$
6130 INPUT "NOVO ITEM ";D$:IF D
$="" THEN 6150
6140 TE$(M)=D$
6150 INPUT "NOVA QUANTIA ";A:IF
 A=0 THEN 6170
6160 IF A<0 OR A>9999999 THEN 6
150 ELSE AM(M)=A
6170 INPUT "NOVA CATEGORIA ";CA
$:IF CA$="" THEN 6200
6180 GOSUB 1180:IF F=0 THEN 617
0
6190 CA(M)=NM
6200 IF CA(M)<>8 THEN GT=GT+AM(
M)
6210 RETURN
7000 CLS: PRINT @69,"VOCE TEM C
ERTEZA (S/N) ?;
7010 A$=INKEY$:IF A$<>"S" AND A
$<>"N" THEN 7010
7020 IF A$="N" THEN RETURN
```

```
 50 LET mn=200: IF PEEK 23733=
 127 THEN  LET mn=100
100 DIM c$(8,16): DIM a(mn):
DIM a$(mn,23)
110 LET u=0: LET v=1
120 FOR n=v TO 8: READ c$(n):
NEXT n
130 POKE 23658,8
140 LET k$=".00": FOR n=v TO 7
145 LET k$=k$+CHR$ 8: NEXT n
190 LET p=2: LET tt=u: LET cr=
u
200 CLS : PRINT  BRIGHT v;
PAPER 2; INK 6;AT 2,1;" M E N
U   P R I N C I P A L "
210 PRINT  BRIGHT v; PAPER 7;
AT 5,4;" 1- ENTRAR REGISTRO  ";
AT 7,4;" 2- VER REGISTROS    ";
AT 9,4;" 3- GRAVAR EM FITA   ";
AT 11,4;" 4- CARREGAR         "
;AT 13,4;" 5- IMPRIMIR
";AT 15,4;" 6- MUDAR REGISTRO
 ";AT 17,4;" 7- SAIDA
 "
220 PRINT  INK 3; FLASH v;
BRIGHT v;AT 20,4;" - SELECIONE
 OPCAO - "
```

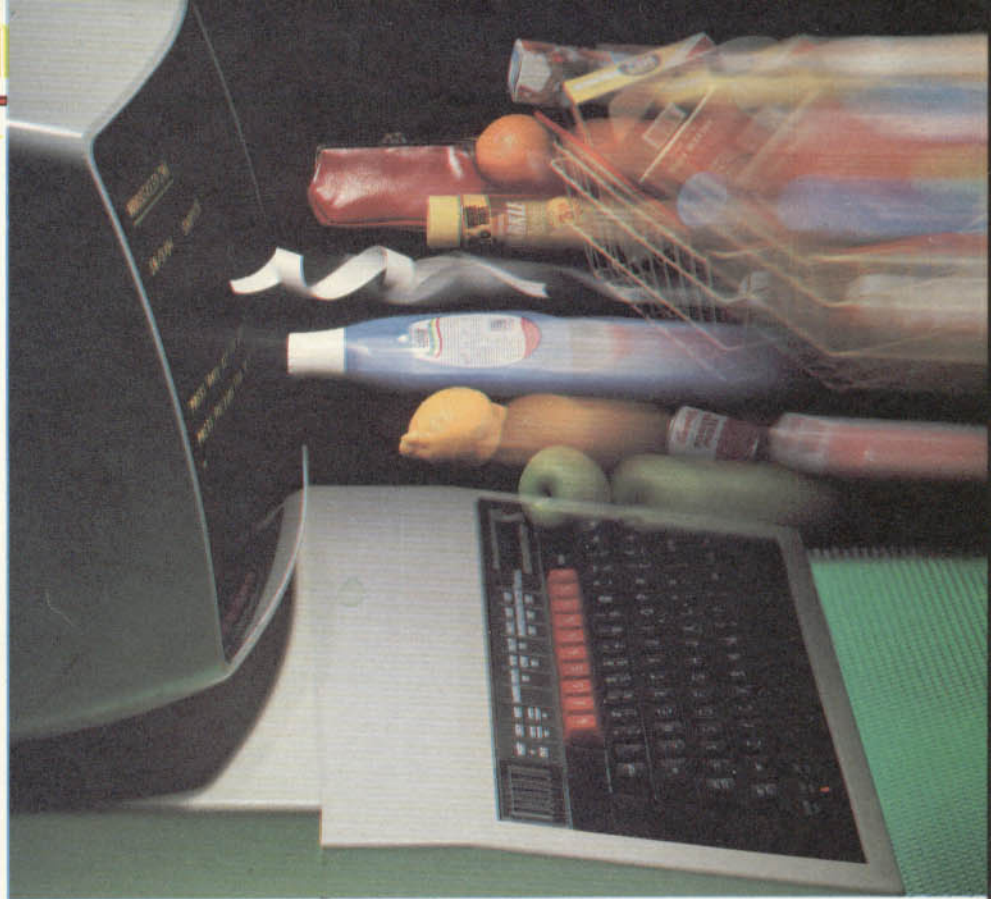
```
 230 IF INKEY$="" THEN  GOTO
  230
 240 LET z$=INKEY$: IF z$<"1"
 OR z$>"7" THEN  GOTO 230
 250 CLS : GOSUB 1000*VAL z$
 260 GOTO 200
1000 LET c=u
1005 LET c=c+v: IF c=mn+v THEN
 RETURN
1006 IF a$(c,v)=" " THEN  GOTO
1010
1007 GOTO 1005
1010 PRINT AT u,u; BRIGHT v; PA
PER 2; INK 7;" DATA      ITEM
   QUANTIA CAT "
1015 IF c=mn+v THEN  RETURN
1020 INPUT "Digite a data"; LIN
E a$(c,2 TO 9): IF a$(c,2)=" "
THEN  RETURN
1030 PRINT TAB u;a$(c,2 TO 9);
1040 INPUT "Digite o item "; LI
NE a$(c,10 TO 23): IF a$(c,10)=
" " THEN  GOTO 1040
1050 PRINT TAB 9;a$(c,10 TO 21)
;
1060 INPUT "Quantia ";a(c): IF
a(c)=u THEN  GOTO 1060
1070 LET vv=a(c)*100: LET v$=ST
R$ vv: PRINT TAB 27-LEN v$;a(c)
;
1080 INPUT "Categoria "; LINE f
$: IF f$="" THEN  GOTO 1080
1090 FOR n=v TO 8: IF f$=c$(n,v
 TO LEN f$) THEN  GOTO 1130
1100 NEXT n: GOTO 1080
1130 IF n=8 THEN  LET cr=cr+a(c
)
1140 IF n<>8 THEN  LET tt=tt+a(
c)
1150 PRINT TAB 29;c$(n,v TO 3)
1160 LET a$(c,v)=CHR$ (48+n)
1200 LET c=c+v: GOTO 1015
2000 FOR n=v TO 8: PRINT  PAPER
 v; INK 7;AT n*2,4;" ";n;"- ";c
$(n): NEXT n
2010 PRINT  FLASH v; INK 2;AT 1
9,3;" Selecione categoria(1 a 8
) "
2020 IF INKEY$="" THEN  GOTO 20
20
2030 LET z$=INKEY$: IF z$<"1" O
R z$>"8" THEN  GOTO 2020
2040 LET t=u: LET c=u
2050 CLS : PRINT #p; PAPER 6; B
RIGHT v;TAB 10;c$(VAL z$);TAB 3
1;" "
2055 LET c=c+v: IF c=mn THEN  G
OTO 2500
2060 IF a$(c,v)=" " THEN  GOTO
2500
2070 IF a$(c,v)<>z$ THEN  GOTO
2055
2080 PRINT #p;a$(c,2 TO 9);TAB
10;a$(c,10 TO 23);
2090 LET am=a(c)*100: LET n$=ST
R$ am: PRINT #p;TAB 29;k$;TAB 3
1-LEN n$;a(c)
2100 LET t=t+a(c)
2110 GOTO 2055
2500 PRINT #p;TAB 25;"-------":
 LET tx=t*100: LET n$=STR$ tx
2505 PRINT #p;TAB 12;"TOTAL- ";
TAB 29;k$;TAB 31-LEN n$;t
2510 IF z$<>"8" THEN  GOTO 2590
2520 LET tz=tt*100: LET n$=STR$
 tz: PRINT '#p;TAB 4;"DESPESA T
OTAL- ";TAB 29;k$;TAB 31-LEN n$
;tt
2530 LET ba=(t-tt)*100: LET n$=
STR$ ba: PRINT '#p;TAB 10;"BALA
NCO- ";TAB 29;k$;TAB 31-LEN n$;
ba/100
2590 PRINT  PAPER 2; INK 7'"Pre
ssione qualquer tecla para   co
ntinuar"
2600 PAUSE u: IF PEEK 23560=13
THEN  RETURN
2610 CLS : GOTO 2000
3000 GOSUB 8000: IF re=v THEN
RETURN
3010 PRINT  PAPER 6;AT 10,u;" D
igite o nome do arquivo data "
3015 INPUT  LINE w$: IF LEN w$>
10 OR LEN w$<v THEN  GOTO 3010
3020 CLS : SAVE w$ DATA a(): SA
VE w$ DATA a$(): RETURN
4000 GOSUB 8000: IF re=v THEN
RETURN
4010 PRINT  BRIGHT v;AT 10,u;"D
igite o nome dos dados a serem
carregados": INPUT  LINE w$. IF
 LEN w$>10 THEN  GOTO 4010
4020 PRINT  PAPER 3; INK 7;AT 1
0,u;"               Pressione  PLA
Y                "
4030 LOAD w$ DATA a(): LOAD w$
DATA a$()
4040 LET cr=u: LET tt=u: FOR n=
v TO mn: IF a$(n,v)="8" THEN  L
ET cr=cr+a(n)
4050 IF a$(n,1)<>"8" THEN  LET
tt=tt+a(n)
4060 NEXT n: RETURN
5000 PRINT  BRIGHT v;AT 10,u;"
Voce quer imprimir (S/N)? "
5010 PAUSE u: IF INKEY$="" THEN
  GOTO 5010
5020 LET z$=INKEY$
5030 IF z$="n" THEN  LET p=2: R
ETURN
5040 IF z$="s" THEN  LET p=3: R
ETURN
5050 GOTO 5010
6000 LET c=v: IF a(c)=u THEN  R
ETURN
6010 PRINT AT u,u; BRIGHT v; PA
PER (VAL a$(c,v))-v; INK 9;" Nu
mero ";c,c$(VAL a$(c,v))
6015 PRINT  PAPER 2; INK 7;''"D
ATA       ITEM           QUANTIA
": PRINT 'a$(c,2 TO 9);TAB 10;a
$(c,10 TO 23);
6020 LET am=a(c)*100: LET n$=ST
R$ am: PRINT TAB 29;k$;TAB 31-L
EN n$;a(c)
6030 PRINT  PAPER 3; INK 7;AT 2
0,u;" A - Prossegue   Q - Retor
na          EDIT para alterar re
gistro "
6040 PAUSE u
6050 IF INKEY$="q" AND c>v THEN
  LET c=c-v: GOTO 6010
6060 IF INKEY$="a" AND c<>mn TH
EN  LET c=c+v
6070 IF a(c)=u THEN  LET c=c-v
6080 IF PEEK 23560=7 THEN  GOTO
 6100
6090 GOTO 6010
6100 INPUT  BRIGHT v;"Digite a
nova data "; LINE a$(c,2 TO 9):
 IF a$(c,2)=" " THEN  GOTO 6100
6110 PRINT AT 5,u;a$(c,2 TO 9)
6120 INPUT  BRIGHT v;"Digite o
novo item "; LINE a$(c,10 TO 23
): IF a$(c,10)=" " THEN  GOTO 6
120
6130 PRINT AT 5,10;a$(c,10 TO 2
3)
6135 IF a$(c,v)="8" THEN  LET c
r=cr-a(c)
6136 IF a$(c,v)<>"8" THEN. LET
```

```
tt=tt-a(c)
6140 INPUT  BRIGHT v;"Digite no
va quantia ";a(c): IF a(c)=u TH
EN  GOTO 6140
6150 LET am=a(c)*100: LET n$=ST
R$ am: PRINT AT 5,29;k$;TAB 31-
LEN n$;a(c)
6160 INPUT  BRIGHT v;"Digite a
nova categoria "; LINE f$: IF f
$="" THEN  GOTO 6160
6170 FOR n=v TO 8: IF f$=c$(n,v
 TO LEN f$) THEN  GOTO 6190
6180 NEXT n: GOTO 6160
6190 LET a$(c,v)=CHR$ (48+n)
6200 IF n=8 THEN  LET cr=cr+a(c
)
6210 IF n<8 THEN  LET tt=tt+a(c
)
6220 RETURN
7000 GOSUB 8000: IF re=v THEN
RETURN
7010 RAND USR u
8000 PRINT  PAPER 4;AT 10,9;" V
oce tem certeza? "
8010 PAUSE u: LET re=u: IF INKE
Y$<>"s" THEN  LET re=v
8020 RETURN
9000 DATA "MANUTENCAO CASA","LA
ZER","ALUGUEL E TAXAS","VESTUAR
IO","AUTOMOVEL","FERIAS","OUTRO
S","RENDA"
```

```
10 COLOR 15,4,4:KEYOFF:MOTOROFF
:CLEAR5000
20 DIMTE$(200),AM(200),DA$(200)
,CT$(8),CA(200)
30 FORN=1TO8:READCT$(N):NEXT
40 DATA MANUTENÇAO DA CASA,LAZE
R,ALUGUÉIS E TAXAS,ROUPAS,AUTOM
óVEL,FERIAS,MISCELÂNEA,RENDAS
50 U1$="$$##########,":U2$="####
####,"
60 CLS:LOCATE7,2:PRINT"M E N U
  P R I N C I P A L":LOCATE10,6
:PRINT"1:- ENTRAR DADOS":LOCATE
10,8:PRINT"2:- VER DADOS":LOCAT
E10,10:PRINT"3:- GRAVAR NA FITA
"
70 LOCATE10,12:PRINT"4:- CARREG
AR DA FITA":LOCATE10,14:PRINT"5
:- OPÇÃO DE IMPRESSÃO":LOCATE10
,16:PRINT"6:- ALTERAR DADOS":LO
CATE10,18:PRINT"7:- FIM DE PROG
RAMA"
80 LOCATE10,22:PRINT"-ESCOLHA U
MA OPÇÃO-"
90 A$=INKEY$:IFA$<"1"ORA$>"7"TH
EN90
100 ONVAL(A$)GOSUB1000,2000,300
0,4000,5000,6000,7000
110 COLOR 15,4,4:GOTO60
1000 CLS:IFNU>200THENLOCATE10,1
5:PRINT"MEMóRIA CHEIA!":BEEP
1010 GOSUB1160
1020 GOSUB1250:INPUT"DATA ";DA$
(NU)
1030 IFDA$(NU)=""THENRETURN
1040 IFLEN(DA$(NU))>8THEN1020
1050 LOCATE0,L:PRINTDA$(NU)
1060 GOSUB1250:LINEINPUT"ITEM ?
";TE$(NU):IFLEN(TE$(NU))>25THEN
1060
1070 A$=LEFT$(TE$(NU),15):LOCAT
E17-LEN(A$)/2,L:PRINTA$
1080 GOSUB1250:INPUT"QUANTIA ";
A
1090 IFA>9999999#ORA<0THEN1080
1100 LOCATE25,L:PRINTUSINGU2$;A
:AM(NU)=A
1110 GOSUB1250:INPUT"CATEGORIA
";CA$
1120 GOSUB1180:IFF=0THEN1110
1130 CA(NU)=NM:LOCATE36,L:PRINT
LEFT$(CT$(CA(NU)),3)
1140 IFNM<>8THENGT=GT+A
1150 NU=NU+1:L=L+1:IFL=19THEN10
00ELSE1020
1160 L=2:PRINT"  data";TAB(14);
"ítem";TAB(26);"quantia";TAB(36
)"cat";
1170 RETURN
1180 IFVAL(CA$)<>0THEN1230
1190 F=0:FORN=1TO8
1200 IFCA$=LEFT$(CT$(N),LEN(CA$
))THENF=F+1:NM=N
1210 NEXT:IFF>1THENF=0
1220 RETURN
1230 IFVAL(CA$)>8THENF=0:RETURN
1240 NM=VAL(CA$):F=1:RETURN
1250 LOCATE0,20:PRINTSPACE$(79)
;:LOCATE0,20:RETURN
2000 CLS:FORN=1TO8
2010 LOCATE10,2*N+3:PRINTN;":-
";CT$(N)
2020 NEXT
2030 TT=0:LOCATE15,23:PRINT"Qua
l categoria? ";
2040 A$=INKEY$:IFA$<"1"ORA$>"8"
THEN2040
2050 NM=VAL(A$)
2060 IFPT=1THENLPRINTTAB(40-LEN
(CT$(NM))/2);CT$(NM):LPRINTTAB(
10)"  DATA";TAB(28);"ITEM";TAB(
51);"QUANTIA"
2070 GOSUB2280
2080 FORNN=0TONU
2090 IFCA(NN)<>NMTHEN2150
2100 IF PT=1 THEN LPRINTTAB(10)
;DA$(NN);TAB(22);TE$(NN);:LPRIN
TTAB(50)USINGU2$;AM(NN)
2110 LOCATE0,L:PRINTDA$(NN);:A$
=LEFT$(TE$(NN),20):LOCATE10:PRI
NTA$;:LOCATE 30:PRINTUSINGU2$;A
M(NN);:TT=TT+AM(NN)
2120 L=L+1:IF (L=21ANDNM<>8) OR
 (L=19ANDNM=8) THEN LOCATE14,22
:PRINT"scroll? ";:ELSE GOTO 215
0
2130 A$=INKEY$:IFA$=""THEN2130
2140 GOSUB2280
2150 NEXT:IFPT=1ANDNM<>8THENLPR
INT:LPRINT:LPRINTTAB(38);USING"
TOTAL => "+U1$;TT
2160 LOCATE18,22:PRINTUSING"tot
al =>"+U1$;TT
2170 IFNM<>8THEN2250
2180 IFPT=0THEN2220
2190 LPRINT:LPRINT:LPRINTTAB(40
);USING"RENDA TOTAL =>"+U1$;TT
2200 LPRINTTAB(36);USING"DESPES
AS TOTAIS =>"+U1$;GT:LPRINTTAB(
54)"--------------"
2210 LPRINTTAB(44);USING"BALANÇ
O =>"+U1$;TT-GT
2220 LOCATE13,20:PRINTUSING"Ren
da total =>"+U1$;TT
2230 LOCATE9,21:PRINTUSING"Desp
esas totais =>"+U1$;GT
2240 LOCATE17,22:PRINTUSING"Bal
anço =>"+U1$;TT-GT;
2250 A$=INKEY$:IFA$=""THEN2250
2260 IFA$<>CHR$(13)THEN2000
2270 RETURN
2280 L=3:CLS:COLOR 15,NM+2:LOCA
TE20-LEN(CT$(NM))/2:PRINTCT$(NM
)
2290 PRINT"  data";TAB(18)"ítem
";TAB(32)"quantia";
2300 FORN=3TO20
2310 VPOKEN*40+10,22:VPOKEN*40+
```

```
30,22
2320 NEXT:RETURN
3000 CLS:MOTOR:LOCATE0,10:PRINT
"Posicione a fita e pressione <
RETURN>"
3010 IFINKEY$=""THEN3010
3020 MOTOR:PRINT"Tecle <REC/PLA
Y> e pressione <RETURN>"
3030 IFINKEY$=""THEN3030
3040 LOCATE8,15:INPUT"Nome do a
rquivo ";DA$:AR$="CAS:"+DA$
3050 OPEN AR$ FOR OUTPUT AS#1
3060 PRINT#1,NU
3070 FORN=0TONU-1
3080 PRINT#1,DA$(N);",";TE$(N);
",";AM(N),CA(N)
3090 NEXT:CLOSE#1:RETURN
4000 CLS:LOCATE8,10:INPUT"Nome
do arquivo ";DA$:AR$="CAS:"+DA$
4010 MOTOR:LOCATE0,12:PRINT"Pos
icione a fita e tecle <RETURN>"
4020 IFINKEY$=""THEN4020
4030 MOTOR:LOCATE0,14:PRINT"Pre
ssione a tecla <PLAY> e <RETURN
>"
4040 IFINKEY$=""THEN4040
4050 MOTOR:GT=0:OPEN AR$ FOR IN
PUT AS#1
4060 LOCATE8,18:PRINT"Achei ";D
A$
4070 INPUT#1,NU
4080 FORN=0TONU-1
4090 INPUT#1,DA$(N),TE$(N),AM(N
),CA(N)
4100 IFCA(N)<>8THENGT=GT+AM(N)
4110 NEXT:CLOSE#1:RETURN
5000 CLS:LOCATE10,10:PRINT"Impr
essora: "
5010 LOCATE15,12:PRINT"[L]igada
"
5020 LOCATE15,13:PRINT"[D]eslig
ada"
5030 A$=INKEY$:IFA$=""THEN5030
5040 IFA$="L"THENPT=1:RETURN
5050 IFA$="D"THENPT=0ELSE5030
5060 RETURN
6000 IFNU=0THENRETURN
6010 CLS:PRINT"  data";TAB(14);
"ítem";TAB(26);"quantia";TAB(36
);"cat"
6020 LOCATE0,20:PRINT"Pressione
 [=>] para avançar"
6030 LOCATE0,21:PRINT"Pressione
 [<=] para retroceder":PRINT"Ou
 a barra de espaços para editar
":GOTO6080
6040 A$=INKEY$:IFA$=""THEN6040
6050 IFASC(A$)=29ANDM>0THENM=M-
1:GOTO6080
6060 IFASC(A$)=28ANDM<NU-1THENM
=M+1:GOTO6080
6070 IFASC(A$)=32THEN6100ELSE60
40
6080 LOCATE0,3:PRINTDA$(M);TAB(
10);LEFT$(TE$(M),15);:PRINTTAB(
26);USINGU2$;AM(M);:PRINTTAB(36
);LEFT$(CT$(CA(M)),3)
6090 GOTO 6040
6100 IFCA(M)<>8THENGT=GT-AM(M)
6110 D$="":PRINT:INPUT"Data ";D
$:IFD$=""THEN6130
6120 IFLEN(D$)>8THEN6110ELSEDA$
(M)=D$
```

```
6130 D$="":PRINT:INPUT"Item ";D
$:IFD$=""THEN6150
6140 IFLEN(D$)>25THEN6130ELSETE
$(M)=D$
6150 PRINT:INPUT"Quantia ";A:IF
A=0THEN6170
6160 IFA<0ORA>9999999#THEN6150E
LSEAM(M)=A
6170 CA$="":PRINT:INPUT"Categor
ia ";CA$:IFCA$=""THEN6200
6180 GOSUB1180:IFF=0THEN6170
6190 CA(M)=NM
6200 IFCA(M)<>8THENGT=GT+AM(M)
6210 RETURN
7000 CLS:LOCATE10,15:PRINT"Fim
do programa?"
7010 A$=INKEY$:IFA$=""THEN 7010
7020 IFA$<>"S"THENRETURN
```

```
10  REM  ONERR  GOTO 8000
20  DIM TE$(200),QT(200),DT$(20
0),CT$(8),CT(200)
25  FOR N = 1 TO 7: READ K$: NE
XT
30  FOR N = 1 TO 8: READ CT$(N)
: NEXT
35  DATA  ENTRAR DADOS,VER DADO
S,GRAVAR EM DISCO,CARREGAR DO D
ISCO,IMPRIMIR,ALTERAR UM DADO,F
IM DE PROGRAMA
40  DATA    MANUTENCAO DA CASA,L
AZER,ALUGUEIS E TAXAS,ROUPAS,AU
TOMOVEL,FERIAS,MISCELANEA,RENDA
S
50  HOME : RESTORE : INVERSE :
VTAB 2: HTAB 7: PRINT " M E N U
   P R I N C I P A L "
60  FOR N = 1 TO 7: READ K$
70  HTAB 10: VTAB N * 2 + 4: PR
INT N;":- ";K$
80  NEXT
90  VTAB 23: HTAB 20: PRINT " O
PCAO -> ";: GET A$
100  IF A$ < "1" OR A$ > "7" TH
EN 90
110  PRINT A$;: NORMAL : HOME
120  ON  VAL (A$) GOSUB 1000,20
00,3000,4000,5000,6000,7000
130  GOTO 50
1000  IF NU > 200 THEN  VTAB 15
: HTAB 10: PRINT  CHR$ (7);"MEM
ORIA CHEIA!"; CHR$ (7): FOR N =
 1 TO 1000: NEXT : RETURN
1010  GOSUB 1160
1020  GOSUB 1250: INPUT "DATA (
DD/MM/AA): ";DT$(NU)
1030  IF DT$(NU) = "" THEN  RET
URN
1040  IF  LEN (DT$(NU)) > 8 THE
N 1020
1050  VTAB L: PRINT DT$(NU)
1060  GOSUB 1250: INPUT "ITEM:
";TE$(NU): IF  LEN (TE$(NU)) >
25 THEN 1060
1070 A$ =  LEFT$ (TE$(NU),15):
VTAB L: HTAB 17 -  LEN (A$) / 2
: PRINT A$
1080  GOSUB 1250: INPUT "QUANTI
A: ";QT(NU)
1090  IF QT(NU) > 99999999 OR Q
T(NU) < 0 THEN 1080
1100  VTAB L: HTAB 26: PRINT QT
(NU)
1110  GOSUB 1250: INPUT "CATEGO
RIA: ";C$
1120 D$ = C$: GOSUB 9000: IF F
= 0 THEN 1110
1130 CT(NU) = NM: VTAB L: HTAB
```

```
38: PRINT  LEFT$ (CT$(CT(NU)),3
);
1140  IF C <  > 8 THEN GT = GT
+ QT(NU)
1150 NU = NU + 1:L = L + 1: IF
L = 21 THEN 1000
1155  GOTO 1020
1160 L = 3: PRINT "  data"; TAB
( 15);"item"; TAB( 26);"quantia
"; TAB( 38);"cat";
1170  RETURN
1250  HTAB 1: VTAB 23: CALL  -
958: RETURN
2000  HOME : FOR N = 1 TO 8
2010  HTAB 10: VTAB 2 * N + 3
2020  PRINT N;":- ";CT$(N): NEX
T
2030 TT = 0: HTAB 15: VTAB 23:
PRINT "Qual categoria? ";
2040  GET A$: IF A$ < "1" OR A$
 > "8" THEN 2040
2050 NM =  VAL (A$)
2060  IF PT = 1 THEN  PRINT : P
RINT  CHR$ (4);"PR#1": PRINT  C
HR$ (9);"80N": PRINT  SPC( 40 -
  LEN (CT$(NM)) / 2);CT$(NM): P
RINT : PRINT  SPC( 10);"  DATA"
; SPC( 17);"ITEM"; SPC( 18)"QUA
NTIA": PRINT  CHR$ (4);"PR#0"
2070  GOSUB 2280
2080  FOR NN = 0 TO NU
2090  IF CT(NN) <  > NM THEN 21
50
2100  IF PT = 1 THEN  PRINT : P
RINT  CHR$ (4);"PR#1": PRINT  C
HR$ (9);"80N";: PRINT  SPC( 10)
;DT$(NN); SPC( 5 + (8 -  LEN (D
T$(NN))));TE$(NN); SPC( 6 + 25
-  LEN (TE$(NN))); SPC( 7 -  LE
N ( STR$ (QT(NN))));QT(NN): PRI
NT  CHR$ (4);"PR#0"
2110  VTAB L: HTAB 1: PRINT DT$
(NN);:A$ =  LEFT$ (TE$(NN),20):
 HTAB 10: PRINT A$;: HTAB 32 +
(7 -  LEN ( STR$ (QT(NN)))): PR
INT QT(NN);:TT = TT + QT(NN)
2120 L = L + 1: IF (L = 21 AND
NM <  > 8) OR (L = 19 AND NM =
8) THEN  VTAB 23: HTAB 15: PRIN
T "Scroll ?";: GET A$: GOSUB 22
80
2150  NEXT : IF PT = 1 AND NM <
  > 8 THEN  PRINT : PRINT  CHR$
 (4);"PR#1": PRINT  CHR$ (9);"8
0N": PRINT : PRINT : PRINT  SPC
( 40);"TOTAL =>";TT: PRINT  CHR
$ (4);"PR#0"
2160  VTAB 23: HTAB 21: PRINT "
TOTAL =>";TT;
2170  IF NM <  > 8 THEN 2250
2180  IF PT = 0 THEN 2220
2190  PRINT : PRINT  CHR$ (4);"
PR#1": PRINT  CHR$ (9);"80N": P
RINT : PRINT : PRINT  SPC( 40);
"RENDA TOTAL =>"; SPC( 9 -  LEN
 ( STR$ (TT)));TT: PRINT  SPC(
36);"DESPESAS TOTAIS =>"; SPC(
9 -  LEN ( STR$ (GT)));GT
2200  PRINT  SPC( 54);"--------
-"
2210  PRINT  SPC( 44);"BALANCO
=>"; SPC( 9 -  LEN ( STR$ (TT -
GT)));TT - GT: PRINT  CHR$ (4)
;"PR#0"
2220  VTAB 21: HTAB 18: PRINT "
RENDA TOTAL =>";TT
2230  VTAB 22: HTAB 15: PRINT "
DESPESAS TOTAIS =>";GT;
2240  VTAB 23: HTAB 20: PRINT "
BALANCO =>";TT - GT;
2250  GET A$
2260  IF A$ <  >  CHR$ (13) THE
N 2000
2270  RETURN
2280  HOME :L = 4: HTAB 20 -  L
EN (CT$(NM)) / 2: PRINT CT$(NM)
2290  PRINT "  data"; TAB( 18);
"item"; TAB( 32);"quantia";
2300  FOR N = 4 TO 21: VTAB N:
HTAB 9: PRINT "!";: HTAB 31: PR
INT "!";: NEXT
2310  RETURN
3000  HOME : VTAB 15: HTAB 10:
INPUT "Nome do arquivo? ";DA$
3010  IF DA$ = "" THEN  RETURN
3020  PRINT : PRINT  CHR$ (4);"
OPEN";DA$
3030  PRINT  CHR$ (4);"DELETE";
DA$
3040  PRINT  CHR$ (4);"OPEN";DA
$
3050  PRINT  CHR$ (4);"WRITE";D
A$
3060  PRINT NU
3070  FOR N = 0 TO NU - 1: PRIN
T DT$(N); CHR$ (13);TE$(N); CHR
$ (13);QT(N); CHR$ (13);CT(N):
NEXT
3080  PRINT  CHR$ (4);"CLOSE":
RETURN
4000  HOME : VTAB 15: HTAB 10:
INPUT "Nome do arquivo? ";DA$
4010  IF DA$ = "" THEN  RETURN
4015 GT = 0
4020  PRINT : PRINT  CHR$ (4);"
OPEN";DA$
4030  PRINT  CHR$ (4);"READ";DA
$
4040  INPUT NU
4050  FOR N = 0 TO NU - 1
4060  INPUT DT$(N),TE$(N),QT(N)
,CT(N)
4065  IF CT(N) <  > 8 THEN GT =
 GT + QT(N)
4070  NEXT
4080  PRINT  CHR$ (4);"CLOSE"
4090  RETURN
5000  HOME : VTAB 15: HTAB 10:
PRINT "Ligo a impressora? (S/N)
";
5010  GET A$: IF A$ <  > "S" AN
D A$ <  > "N" THEN 5010
5020  PRINT : PRINT  TAB( 20);"
OK!": IF A$ = "N" THEN PT = 0:
RETURN
5030 PT = 1: RETURN
6000  IF NU = 0 THEN  RETURN
6020  VTAB 20: PRINT : PRINT "P
ressione [->] para avancar": PR
INT "Pressione [<-] para retroc
eder": PRINT "Ou a barra de esp
acos para corrigir";: GOTO 6080
6030  GET A$
6040  IF A$ =  CHR$ (32) THEN 6
100
6050  IF A$ =  CHR$ (21) AND M
< NU - 1 THEN M = M + 1: GOTO 6
080
6060  IF A$ =  CHR$ (8) AND M >
 0 THEN M = M - 1: GOTO 6080
6070  GOTO 6030
6080  VTAB 5: HTAB 1: CALL  - 8
68: PRINT "DATA:        ";DT$(M)
6090  VTAB 8: CALL  - 868: PRIN
T "ITEM:        ";TE$(M): VTAB 11
: CALL  - 868: PRINT "QUANTIA:
  ";QT(M): VTAB 14: CALL  - 868
: PRINT "CATEGORIA: ";CT$(CT(M)
)
6095  GOTO 6030
6100  IF CT(M) <  > 8 THEN GT =
 GT - QT(M)
6110  FOR X = 1 TO 4
6120  VTAB X * 3 + 2: HTAB 12:
CALL  - 868: INPUT "";D$
6130  ON X GOTO 6200,6300,6400,
6500
6140  NEXT
6150  IF CT(M) <  > 8 THEN GT =
 GT + QT(M)
6160  RETURN
6200  IF D$ = "" THEN D$ = DT$(
M): GOTO 6600
6210  IF  LEN (D$) > 8 THEN 612
0
6220 DT$(M) = D$: NEXT
6300  IF D$ = "" THEN D$ = TE$(
M): GOTO 6600
6310  IF  LEN (D$) > 25 THEN 61
20
6320 TE$(M) = D$: NEXT
6400  IF  VAL (D$) = 0 THEN D$
=  STR$ (QT(M)): GOTO 6600
6410  IF  VAL (D$) > 9999999 TH
EN 6120
6420 QT(M) =  VAL (D$): NEXT
6500  IF D$ = "" THEN D$ = CT$(
CT(M)): GOTO 6600
6510  GOSUB 9000: IF F = 0 THEN
 6120
6520 CT(M) = NM: GOTO 6140
6600  VTAB 3 * X + 2: HTAB 12:
PRINT D$: NEXT : RETURN
7000  VTAB 15: HTAB 8: PRINT "T
ermino o programa? (S/N)";
7010  GET A$: IF A$ <  > "S" AN
D A$ <  > "N" THEN 7010
7020  IF A$ = "N" THEN  RETURN
7030  END
8000  IF  PEEK (222) <  > 5 THE
N  RESUME
8005  PRINT  CHR$ (4);"CLOSE"
8010  PRINT  CHR$ (4);"DELETE";
DA$
8020  HOME : VTAB 20: HTAB 10:
FLASH : PRINT "ARQUIVO INEXISTE
NTE!"
8030  FOR I = 1 TO 2000: NEXT :
 NORMAL
8040  GOTO 50
9000  IF  VAL (D$) <  > 0 THEN
9030
9010 F = 0: FOR N = 1 TO 8
9020  IF  LEFT$ (D$,3) =  LEFT$
 (CT$(N),3) THEN F = 1:NM = N
9025  NEXT : RETURN
9030  IF  VAL (D$) > 8 THEN F =
 0: RETURN
9040 NM =  VAL (D$):F = 1: RETU
RN
```

FAÇA PROGRAMAS MAIS CURTOS

- USE COMANDOS ABREVIADOS
- COMO ECONOMIZAR MEMÓRIA
- COMANDOS MÚLTIPLOS
- COMO MELHORAR A VELOCIDADE DE UM PROGRAMA

Programas mais curtos ocupam menores espaços na memória do computador, além de serem de execução mais rápida. Aprenda a trabalhar com eles, melhorando o desempenho do seu micro.

Uma das regras básicas para se escrever um programa é torná-lo claro e legível, usando linhas curtas com comentários e espaços em branco e, se possível, com nomes longos de variáveis.

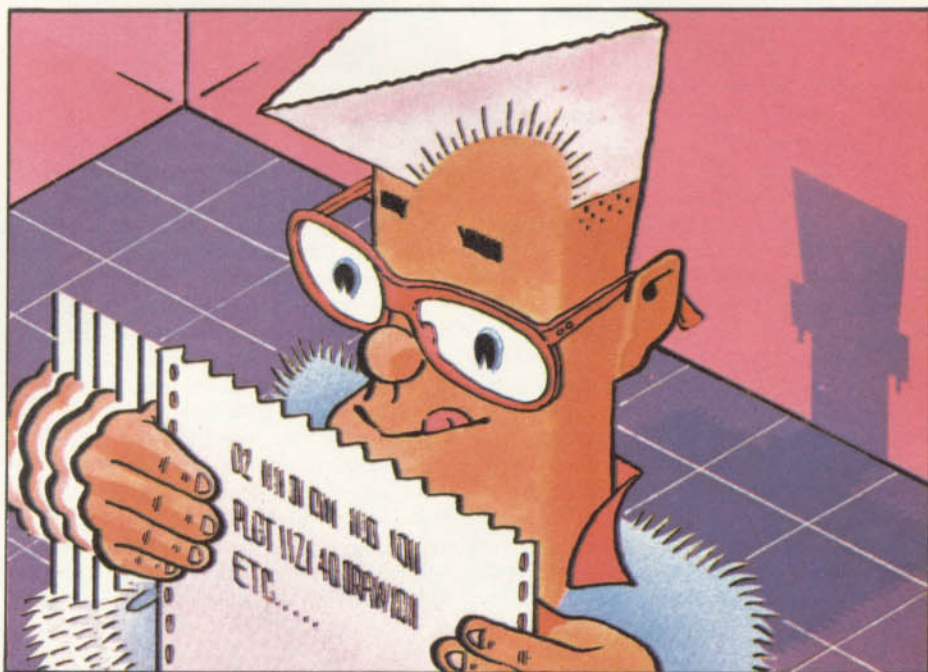
Evitar a digitação de muitos caracteres é uma das exigências mais importantes quando se pretende, por exemplo, copiar a longa listagem de uma revista. Neste caso, não há necessidade de digitar todas as declarações **REM** do programa. Afinal, sempre é possível consultar a revista mais uma vez, caso a finalidade de alguma seção do programa tenha sido esquecida.

Outra maneira de reduzir o trabalho de digitação na entrada de um programa é omitir palavras-chaves opcionais do BASIC, embora nem todos os micros admitam esse tipo de omissão (é o caso do Sinclair). Mas em outros computadores você pode deixar de lado todos os **LET** e **THEN**, que são declarações opcionais, ou mesmo — como no caso de uma linha de programa que contenha **IF...THEN GOTO...** — omitir o **THEN** ou o **GOTO** (a eliminação de ambas as partes, porém, não é possível).

Alguns micros permitem que se entrem palavras-chaves de forma abreviada, como ? em vez de **PRINT**, e ' (apóstrofo), para substituir **REM**, nos micros das linhas TRS-80 e TRS-Color.

Quando o programa for listado ou impresso, as abreviações aparecerão por extenso (com exceção do apóstrofo). Assim, embora isso poupe tempo de digitação, não leva a nenhuma economia de memória, e nem faz com que o seu programa rode mais rápido.

COMO ECONOMIZAR MEMÓRIA

Outra razão para se encurtar um programa é economizar espaço da memória. Essa economia é decisiva em programas muito longos.

Uma das formas de economizar bytes extras é utilizar nomes de variáveis com uma letra apenas. Esse procedimento poupa não só tempo de digitação, como também espaço de memória. Mas é sempre bom manter uma lista sobre o que faz cada variável, já que fica mais difícil usar códigos mnemônicos para variáveis com apenas uma letra.

Outra maneira de se encurtar um programa é omitir espaços em branco entre comandos, variáveis e operadores. Isso não se aplica ao ZX-81 e ao Spectrum (nesses micros, os espaços aparecem automaticamente). Em outros micros a omissão de espaços é uma medida extrema, pois ela torna muito difícil a leitura do programa. Uma regra, contudo, deve ser sempre observada: você deve deixar um espaço entre uma variável e o início de uma palavra-chave. Assim, uma linha como essa:

```
IF A = B AND B = C THEN PRINT
"OK"
```

pode se tornar mais curta desse modo:

```
IFA=B ANDB=C THENPRINT"OK"
```

Mas, se você escreveu:

```
IF A=BAND B=C THENPRINT"OK"
```

o computador pensará que existe uma variável chamada **BAND**, e, como ela não existe, poderá ocorrer um erro durante a execução do programa (ou, em alguns micros, uma mensagem de erro).

Um terceiro modo de economizar memória consiste em combinar várias declarações em uma única linha. Na maioria dos computadores descritos aqui (linhas TRS-80, TRS-Color, MSX, Sinclair Spectrum, Apple II e TK-2000), o sinal de separação entre os comandos contidos em uma mesma linha é : (dois pontos). Apenas nos micros compatíveis com a linha ZX-81 isso não é possível. Mas esse recurso torna os programas mais difíceis de serem entendidos.

A maioria dos métodos aqui indicados tende a acelerar ligeiramente a velocidade de execução de um programa. Outro truque para se nuclear a execução de laços **FOR...NEXT** é omitir a variável que vem após o **NEXT** (isso não é permitido em alguns micros). Se o programa tiver vários laços aninhados, terminando na mesma linha, podemos colocar um único **NEXT**; por exemplo: **NEXT A, B, C** (apenas certos micros aceitam esta sintaxe).

ABAIXO DE ZERO

Diretamente relacionados com o funcionamento interno do computador, binários e hexadecimais apresentam problemas quando é preciso representar números negativos.

Durante a programação em código de máquina, é comum nos depararmos com números negativos, como por exemplo quando, numa programação de jogos, somos levados a movimentar figuras em determinadas direções ao longo da tela.

Como todos os outros, esses números devem ser codificados em bytes de oito bits, pois os computadores com unidades de memória desse tipo não contam com outros recursos para armazenagem de algarismos. Isto, porém, levanta um problema: como você já viu, um byte pode representar qualquer número de 0 a 255, ou 00000000 a 11111111, em binário. Mas isso esgota todas as possibilidades do binário de oito bits, já que não existe espaço para sinais de mais (+) ou menos (–), e nenhuma maneira pela qual eles possam ser representados nessa gama de valores. Em aritmética simples, se você subtrair 1 de 0 obterá −1. Agora, tente o mesmo cálculo em binário com oito bits:

```
00000000
      −1
--------
11111111
```

Esse resultado levará você, ao chegar ao oitavo bit à esquerda, a pedir "emprestado" um do próximo lugar à esquerda; mas quando o número binário é limitado a oito bits não existe nada à esquerda a ser emprestado. Do mesmo modo, se você subtrair outro 1 (para obter −2 em aritmética simples), obterá 11111110. Mas se 11111111 em binário equivale a 255 em decimal, então 11111110 deverá ser 254!

Imagine agora, por exemplo, o que poderia acontecer em decimal se você não tivesse mais que três dígitos ou "colunas" para colocar seus números. Veja o que acontece se você tentar acrescentar 999 a 100, quando tais limitações são impostas:

```
  100
 +999
-----
(1)099
```

O número um, na unidade de milhar, teve que ser colocado entre parênteses. Em nossa aritmética de três dígitos, simplesmente não existe espaço para ele. Assim, o resultado dessa adição, em um sistema de três dígitos, será 99, que é exatamente o que você obterá se subtrair 1 de 100!

Do mesmo modo, em um sistema decimal de três dígitos, o resultado da soma de 998 mais 100 seria o mesmo da subtração de 100 menos 2. E subtrair 998 de 100 seria o mesmo que adicionar 2.

Recapitulando: a mesma série de algarismos em um byte de oito bits pode representar um número negativo e um positivo. Qualquer operação torna-se confusa e difícil diante disso.

O que você pode fazer em relação a isso? Bem, na maioria das aplicações em computadores domésticos, não há com que se preocupar: os endereços de memória e códigos de operação, ambos representados em binário, serão sempre considerados positivos. As únicas vezes em que você encontrará números negativos serão com dados ou com os chamados *saltos relativos,* os quais são códigos de máquina aproximadamente equivalentes às declarações **GOTO** do BASIC.

COMO "VIRAR" OS BITS

O processo utilizado em binário para se obter, a partir de um número positivo, o seu valor negativo, é conhecido como complemento de 2. Na realidade, essa técnica não tem uma base teórica muito bem fundamentada; o importante é que ela funciona.

Para se obter o valor negativo de um número binário, é necessário "virar" os bits e acrescentar 1. "Virar os bits" significa transformar os bits 1 em 0 e aqueles que têm valor 0, em 1.

No programa seguinte mostramos como isso funciona. Note que, quando o binário for limitado a oito dígitos, seu equivalente em hexadecimal preencherá exatamente dois dígitos. Isso significa que o hexadecimal equivalente ao complemento de 2 também atuará como o negativo.

T

```
10 CLS
20 PRINT@8,"NUMEROS NEGATIVOS";
30 PRINT@40,STRING$(16,CHR$(131
));
40 PRINT@65,"DEC"TAB(15)"BIN"TA
B(28)"HEX"
50 PRINT@226,"+"TAB(29)"+";
60 PRINT@361,"COMPLEMENTO DE 2"
70 PRINT@483,"0     0 0 0 0  0 0
 0 0      00";
80 FOR J=1474 TO 1502:POKE J,13
1:NEXT
90 FOR J=1 TO 7
100 FOR K=1 TO 24
110 POKE 1123+K+32*J,175
120 NEXT K,J
130 PRINT@174,"BITS";
140 PRINT@313,"+1";
150 AT=AT AND 255
160 T=AT: IF T=128 THEN SOUND 3
0,2
170 LN=3:GOSUB 280: GOSUB 310
180 T=T+256*(T>127):GOSUB 340
190 T=255-T:LN=7:GOSUB 280
200 T=T+1:T=T AND 255:LN=13:GOS
UB 280: GOSUB 310
210 T=T+256*(T>128):GOSUB 340
220 IN$=INKEY$:IF IN$<>"B" AND
IN$<>" " AND IN$<>CHR$(13) THEN
 220
230 IF IN$="B" THEN AT=AT-1:GOT
O 150
240 IF IN$=" " THEN AT=AT+1:GOT
O 150
250 PRINT @384,;:INPUT AT
260 PRINT @384,"      ";
270 GOTO 150
280 FOR X=0 TO 7
290 IF-(T AND 2^X) THEN PRINT @
LN*32+23-X*2+(X>3),"1";ELSE PRI
NT @LN*32+23-X*2+(X>3),"0";
300 NEXT:RETURN
310 IF T<16 THEN A$="0" ELSE A$
=""
320 PRINT @LN*32+29,A$+HEX$(T);
330 RETURN
340 PRINT @32*LN,MID$("     "+STR
$(T),LEN(STR$(T)));
350 RETURN
```

```
10 PRINT AT 0,7;"NUMEROS NEGA
TIVOS"
20 PRINT AT 2,1;"DEC";TAB 14;
"BIN";TAB 28;"HEX"
30 LET A$="
"
40 FOR N=7 TO 13
50 PRINT AT N,6; INVERSE 1;A$
60 NEXT N
70 PRINT AT 8,14;"BITS";
```

- QUANDO SÃO NECESSÁRIOS NÚMEROS NEGATIVOS
- A CONVERSÃO DE NÚMEROS NEGATIVOS
- A CONVENÇÃO DO SINAL

```
BRIGHT 1;AT 12,21;"+1"
80 PRINT AT 10,3;"+";AT 10,30
;"+"
 90 PRINT AT 15,7;"COMPLEMENTO
 DE 2"
 95 LET C=0
100 LET DD=-C: DIM A(8)
110 PRINT AT 4,0;"      ";AT 4,4
-LEN STR$ C;C
115 POKE 23608,C: LET E=PEEK
23608: LET Z=E: GOSUB 300:
PRINT AT 4,29;A$
120 PRINT AT 17,0;"      ";AT 17
,4-LEN STR$ DD;DD
130 LET D=128: LET CC=E
140 FOR N=1 TO 8: LET A(N)=0
150 IF CC-D>=0 THEN  LET A(N)=
1: LET CC=CC-D
160 PRINT  BRIGHT 1;AT 4,6+2*N
;A(N);AT 10,6+2*N;1-A(N)
170 LET D=D/2: NEXT N
180 POKE 23608,DD: LET DD=PEEK
23608: LET D=128
185 LET Z=DD: GOSUB 300: PRINT
AT 17,29;A$
190 FOR N=1 TO 8: LET B=0: IF
DD-D>=0 THEN  LET B=1: LET DD=
DD-D
200 PRINT  BRIGHT 1;AT 17,6+2*
N;B: LET D=D/2: NEXT N
210 PRINT AT 18,0;"----  -----
--------------  ----"
220 PRINT AT 19,3;"0     0 0 0
0 0 0 0 0       00"
230 IF INKEY$="" THEN  GOTO
 230
240 LET A$=INKEY$: IF A$=" "
THEN  LET C=C+1: IF C=128 THEN
 LET C=-128: SOUND 1,1
250 IF A$="B" OR A$="b" THEN
LET C=C-1: IF C=-129 THEN  LET
C=127: SOUND 1,1
260 IF A$<>" " AND A$<>"B" AND
A$<>"b" THEN  INPUT "?";C
270 GOTO 100
```

COMO CONTAR EM COMPUTÊS

Se quisermos contar qualquer coisa em um computador, devemos começar sempre do zero e avançar daí para cima. Por exemplo, as locações de memória são numeradas em hexadecimal, de 0000 a FFFF.

O mesmo acontece com os bits em um byte. Quando numeramos os bits, começamos a partir da direita, com o bit número 0, e aumentamos a contagem à medida que nos deslocamos para a esquerda.

Assim, o dígito binário da extrema direita é chamado de bit 0. O que está imediatamente à sua esquerda é chamado de bit 1, o seguinte de bit 2 e assim por diante, até o bit 7 — aquele que está na extrema esquerda de um byte.

Assim como em política, os termos "à direita" e "à esquerda" podem, em certos casos, parecer um pouco confusos, especialmente quando lidamos com números de dois bytes que podem ser armazenados de cima para baixo, ou de baixo para cima, dependendo de como o computador opera.

Em "computês" correto (ou seja, na linguagem universal dos computadores), esses números são chamados de *valor mais significativo* — isto é, com o endereço mais alto — e *valor menos significativo*.

Dessa forma, se você estiver armazenando um endereço — 3D8E, por exemplo — 3D é o byte mais significativo, enquanto 8E é o byte menos significativo.

Os computadores das linhas Sinclair, TRS-80 e MSX colocam o byte menor ou menos significativo — 8E, no exemplo acima — na locação de memória mais baixa. O byte alto, que vale 100 em hexa (ou 256 em decimal), 3D neste caso, é colocado com o endereço mais alto na locação de memória (isto é, o endereço de memória que recebeu 8E, mais 1).

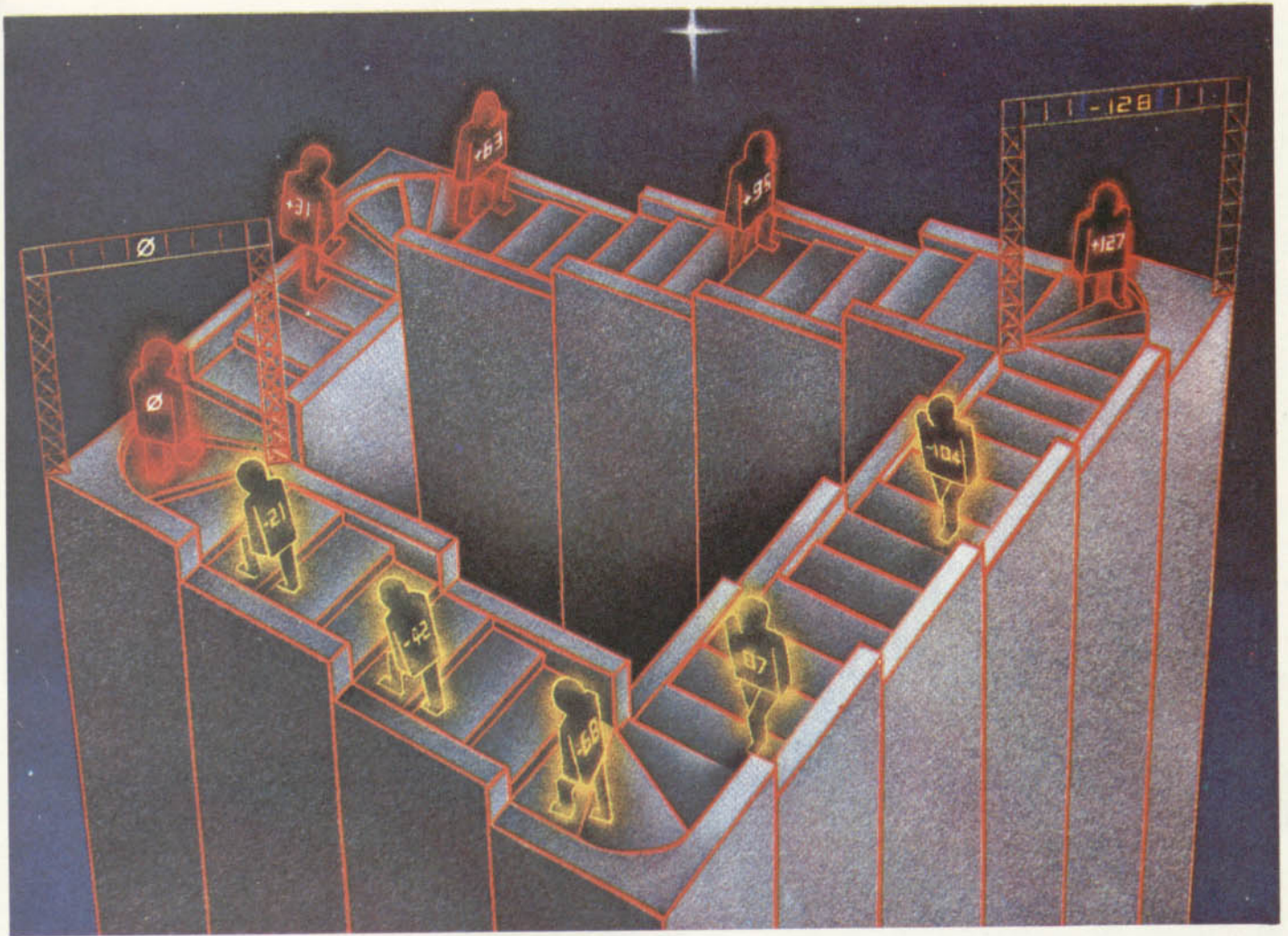
Existe uma única exceção para esta convenção de baixo/alto: os números da linha BASIC são armazenados em forma inversa.

O TRS-Color, por sua vez, armazena todos os números de dois bytes na locação de memória mais baixa com o byte alto, e armazena o byte baixo na locação de memória mais alta.

A mesma terminologia se aplica aos bits. O bit mais significativo no número binário 01010101, o bit 7, é 0, e o bit menos significativo, o bit 0, é o 1.

```
300 LET ZA=INT (Z/16): LET ZB=
Z-(16*ZA)
310 LET ZA=ZA+48: IF ZA>57
THEN  LET ZA=ZA+7
320 LET ZB=ZB+48: IF ZB>57
THEN  LET ZB=ZB+7
330 LET A$=CHR$ ZA: LET A$=A$+
CHR$ ZB: RETURN
```

```
 10 PRINT AT 0,7;"NUMEROS NEGAT
IVOS"
 20 PRINT AT 21,1;"DEC";TAB 14;
"BIN";TAB 28;"HEX"
 30 LET A$="
"
 40 FOR N=7 TO 13
 50 PRINT AT N,6;A$
 60 NEXT N
 70 PRINT AT 8,14;"BITS";AT 12,
21;"+1"
 80 PRINT AT 10,3;"+";AT 10,30;
"+"
 90 PRINT AT 15,8;"COMPLEMENTO
DE 2"
 95 LET C=0
100 LET DD=-C
105 DIM A(8)
110 PRINT AT 4,0;"     ";AT 4,4-
LEN STR$ C;C
115 POKE 16507,C
116 LET E=PEEK 16507
117 LET Z=E
118 GOSUB 300
119 PRINT AT 4,29;A$
120 PRINT AT 17,0;"     ";AT 17,
4-LEN STR$ DD;DD
130 LET D=128
135 LET CC=E
140 FOR N=1 TO 8
145 LET A(N)=0
150 IF CC-D>=0 THEN LET A(N)=1
155 IF CC-D>=0 THEN LET CC=CC-D
160 PRINT AT 4,6+2*N;A(N);AT 10
,6+2*N;1-A(N)
170 LET D=D/2
175 NEXT N
180 POKE 16507,DD
181 LET DD=PEEK 16507
182 LET D=128
185 LET Z=DD
186 GOSUB 300
187 PRINT AT 17,29;A$
190 FOR N=1 TO 8
191 LET B=0
192 IF DD-D>=0 THEN LET B=1
193 IF DD-D>=0 THEN LET DD=DD-D
200 PRINT AT 17,6+2*N;B
205 LET D=D/2
207 NEXT N
210 PRINT AT 18,0;"----  ------
-------------  ----"
220 PRINT AT 19,3;"0     0 0 0 0
0 0 0 0      00"
230 IF INKEY$="" THEN GOTO 230
240 LET A$=INKEY$
242 IF A$="F" THEN LET C=C+1
244 IF A$="F" AND C=128 THEN
LET C=128
250 IF A$="B" THEN LET C=C-1
255 IF A$="B" AND C=-129 THEN
LET C=127
260 IF A$<>"F" AND A$<>"B" THEN
INPUT C
270 GOTO 100
300 LET ZA=INT (Z/16)
305 LET ZB=Z-(16*ZA)
310 LET ZA=ZA+28
320 LET ZB=ZB+28
330 LET A$=CHR$ ZA
340 LET A$=A$+CHR$ ZB
350 RETURN
```

```
10 SCREEN 1:KEY OFF
14 VPOKE BASE(6)+27,166
20 LOCATE 6,0:PRINT "Números ne
gativos";
30 LOCATE 6,1:PRINT STRING$(17,
223);
40 LOCATE 3,3:PRINT "DEC";TAB(1
3);"BIN";TAB(24);"HEX";
50 LOCATE 1,10:PRINT "+";TAB(27
);"+";
60 LOCATE 6,17:PRINT "Complemen
to de 2";
70 LOCATE 0,21:PRINT "    0  0 0
 0 0  0 0 0 0    00 ";
80 FOR J=2 TO 30
84 VPOKE BASE(5)+J+19*32,223
88 NEXT
90 FOR J=1 TO 7
100 FOR K=1 TO 24
110 VPOKE BASE(5)+195+K+32*J,21
9
120 NEXT K,J
130 LOCATE 12,11:PRINT "BITS";
140 LOCATE 23,12:PRINT "+1"
150 AT=AT AND 255
160 T=AT
165 IF T=128 THEN SOUND 7,56:SO
UND 8,15:SOUND 1,1:FORI=1 TO 50
:NEXT:SOUND 8,0
170 LN=3:GOSUB 280
175 GOSUB 310
180 T=T+256*(T>127)
185 GOSUB 340
190 T=255-T:LN=7:GOSUB 280
200 T=T+1:T=T AND 255:LN=13
205 GOSUB 280:GOSUB 310
210 T=T+256*(T>128)
215 GOSUB 340
220 IN$=INKEY$:IF IN$<>"B" AND
IN$<>" " AND IN$<>CHR$(13) THEN
 220
230 IF IN$="B" THEN AT=AT-1:GOT
O 150
240 IF IN$=" " THEN AT=AT+1:GOT
O 150
250 LOCATE 0,20:INPUT AT
260 LOCATE 0,20:PRINT "        ";
270 GOTO 150
280 FOR X=0 TO 7
290 IF -(T AND 2^X) THEN VPOKE
BASE(5)+LN*32+87-X*2+(X>3),ASC(
"1") ELSE VPOKE BASE(5)+LN*32+8
7-X*2+(X>3),ASC("0")
300 NEXT:RETURN
310 IF T<16 THEN A$="0" ELSE A$
=""
320 LOCATE 25,LN+2:PRINT A$+HEX
$(T);
```

```
330 RETURN
340 LOCATE 0,LN+2:PRINT MID$("
  "+STR$(T),LEN(STR$(T)));
350 RETURN
```

Um número binário ou hexadecimal pode, na maior parte das vezes, atuar perfeitamente bem, seja ele um número positivo ou negativo. Mas, às vezes, você precisa saber se um número é positivo ou negativo.

Quando se quer que um programa dê um salto, em código de máquina, é preciso especificar quantos bytes deve ter o salto a ser dado pelo computador: com um número positivo, no caso do salto para a frente; com um número negativo, para saltar para trás.

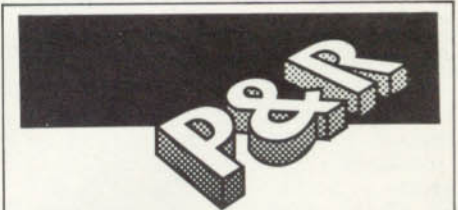
O que o computador faz é olhar para o primeiro bit do número binário e decidir se o número é negativo ou positivo. Se o primeiro bit for 1, o computador o tomará como negativo. Se for 0, ele será considerado como positivo.

Conhecido como *convenção de sinal* esse processo indica que, ao invés de tomar números binários de oito bits para

Na convenção de sinal, o número 128 (ou 10000000, em binário) atua como seu próprio negativo e o computador recomeça a contar de forma ascendente.

representar a série de 0 a 255, o computador a trata como −128 a +127.

No programa de conversão de complemento de 2, você notará que o computador se atrapalha quando chega ao 128. Isso se deve a que o 128, ou seja, 10000000 em binário, ou 80 em hexa, atua como o seu próprio negativo (você pode verificar por si mesmo: −10000000 + 10000000 = (1)00000000 ou 0 em binário de oito bits, do mesmo modo que 80 + 80 = (1)00 ou 0 em hexa de dois dígitos. E 128 − 128 = 0 em decimal).

Portanto, qual deles é positivo ou negativo? O 10000000 tem o 1 em seu primeiro bit; o computador o trata como um número negativo: −128. Por outro lado, o 0 — ou 00000000 — tem o 0 em seu primeiro bit; desse modo, o computador o trata como positivo.

Quando posso utilizar o sistema decimal codificado em binário (BCD)?

O BCD é empregado para imprimir números na tela, ou transmiti-los rapidamente. Para codificar um decimal de dois dígitos em um byte em binário, colocamos o dígito à direita nos últimos quatro bits significativos (um " meio byte") e o dígito à esquerda dentro do meio byte mais significativo.

Existem dezesseis dígitos que podem ser codificados em quatro bits binários. O BCD é um número hexadecimal sem as letras A, B, C, D, E e F. O único problema surge quando obtemos um número maior que 9 em um dos meios bytes.

ORDEM E LIMPEZA NO VÍDEO

- COMO UTILIZAR **TAB** COM **PRINT**
- SINAIS DE PONTUAÇÃO
- COMO MELHORAR A EXIBIÇÃO EM TELA

Listas, tabelas e instruções são mais compreensíveis quando distribuídas pelo vídeo de forma clara e ordenada. Use comandos do BASIC para isso.

Existem muitas maneiras de se dispor textos na tela. Mas é importante organizar bem a exibição, especialmente se você está escrevendo um programa que outras pessoas utilizarão.

Cada computador possui uma maneira própria de posicionar o texto na tela. Algumas funções de posicionamento são padronizadas para todas as variantes do BASIC, independentemente da marca do computador (exemplo: a função **TAB** e os sinais de pontuação — vírgulas e pontos e vírgulas — utilizados para separar os elementos de uma declaração **PRINT**). Entretanto, há outros comandos (associados ou não ao **PRINT**) destinados a posicionar o cursor na tela por meio do endereçamento por linhas e colunas e que variam amplamente entre os diversos computadores:

Comando	Microcomputador
PRINT AT	Sinclair
PRINT @	TRS
LOCATE	MSX
HTAB e **VTAB**	Apple

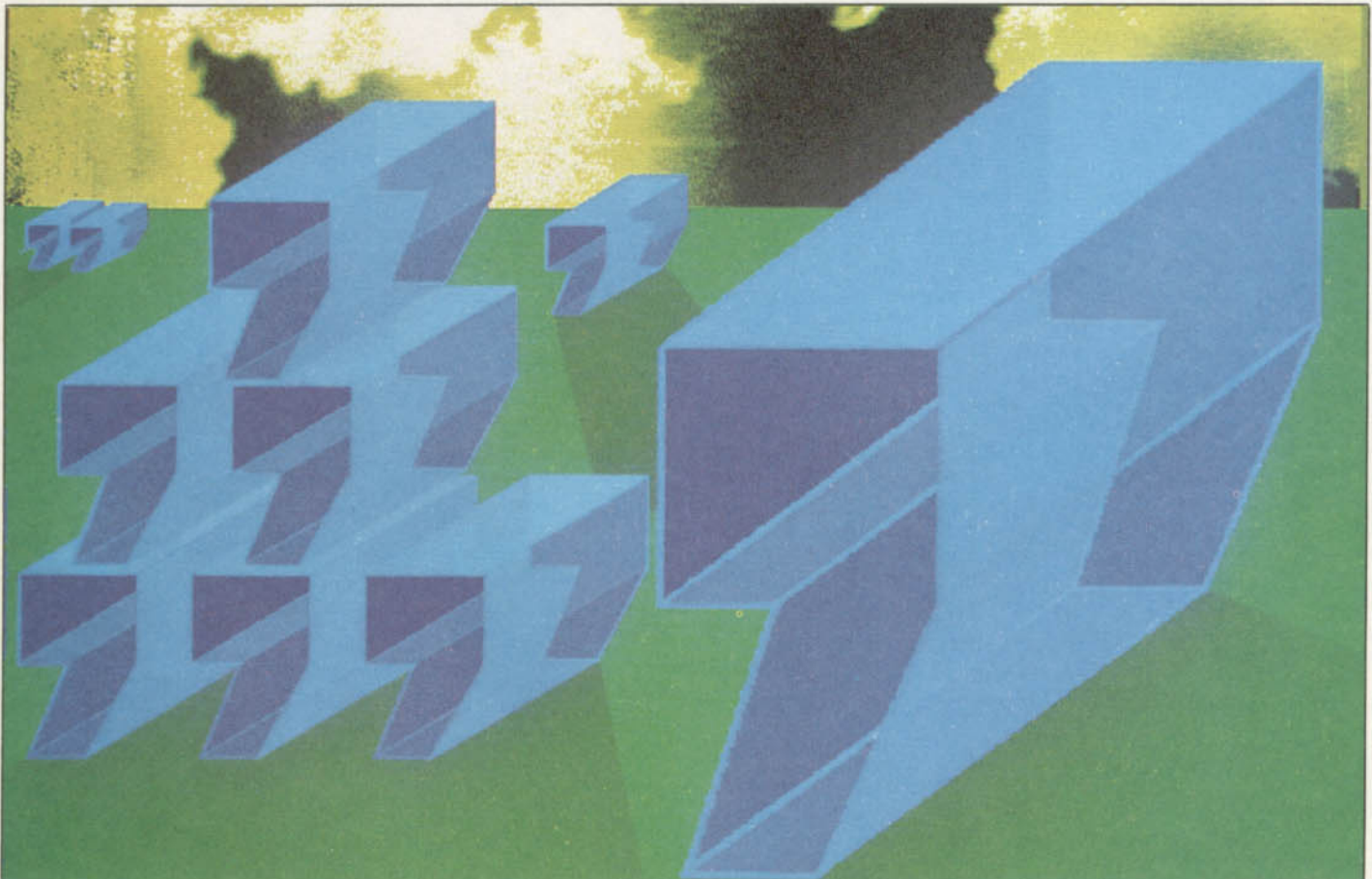
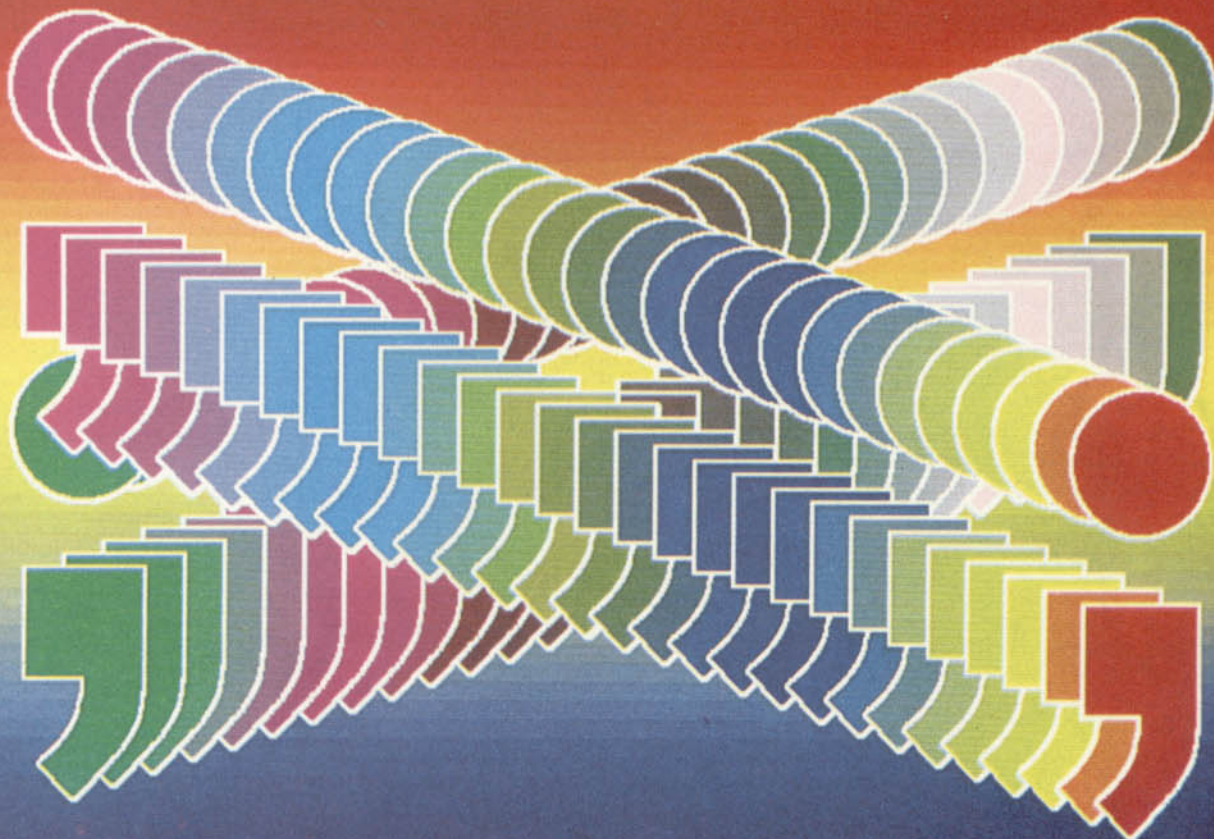
```
PRINT "BOM DIA"
```

imprimirá a mensagem na próxima linha em branco, a partir da margem esquerda da tela.

Entretanto, existem meios para dizer ao computador como imprimir a mensagem em uma posição diferente:

```
PRINT TAB(20)"BOA TARDE"
```

A função **TAB** é usada em conjunto com um **PRINT**. O número entre parênteses especifica a coluna a partir da qual o texto (ou número) que se segue será impresso. Pode-se colocar também uma expressão numérica ou uma variável.

O resultado da expressão numérica ou o número colocado como parâmetro podem ser fracionários, mas o computador usará apenas a sua parte inteira (por exemplo, **TAB (23, 2197)** será aproximado para **23**). O número resultante deve ficar entre 0 e o número máximo de colunas por linha do computador. É possível também posicionar cada palavra separadamente. Modifique a linha para:

```
PRINT TAB(20)"BOA" TAB(30)"
TARDE"
```

A primeira palavra é impressa a partir da coluna 20, e a segunda, a partir da coluna 30 (**TAB** negativo não funciona, ou seja, não se pode recuar posições na linha de impressão).

Não existe espaço em branco entre a palavra-chave **TAB** e o primeiro parêntese. Alguns micros exigem ponto e vírgula entre o **TAB** e o elemento que o segue:

```
PRINT TAB(20);" BOA";
TAB(30);"TARDE"
```

SINAIS DE PONTUAÇÃO

A função **TAB** indica a coluna, numa linha de impressão na tela, a partir da qual o texto será exibido.

O comando **PRINT** emprega vírgula e ponto e vírgula (e também apóstrofo, na linha Sinclair Spectrum), para determinar o espaço entre os elementos a serem impressos.

Quando dois elementos de uma linha **PRINT** são separados por um ponto e vírgula, o computador não concede espaço entre eles e:

```
PRINT "TOCA";"DISCOS"
```

imprimirá TOCADISCOS, sem separação. Se os elementos de **PRINT** forem números, e não cordões alfanuméricos, o resultado dependerá do tipo de micro. Alguns colocarão um espaço em branco antes do número a ser impresso. Assim,

```
PRINT "SOMA";25
```

mostrará o resultado: SOMA 25.

No Sinclair, entretanto, o resultado dessa instrução seria: SOMA25.

A vírgula funciona como a função **TAB**. A cada vírgula encontrada, o programa coloca a próxima informação a ser impressa numa posição distante dez colunas (ou dezesseis, no Sinclair) à direita da última posição inicial. Assim,

```
PRINT "TOCA","DISCOS"
```

produzirá:

TOCA DISCOS.

Em muitos casos, especialmente quando há colunas de números ou de palavras, a informação pode ser posicionada muito mais facilmente e com muito menos esforço de sua parte, exigindo o emprego da função **TAB** apenas de vez em quando. Digite e execute as linhas a seguir, para ter uma idéia de como isso funciona:

```
10 PRINT "0123456789012345667 8
901234567890"
20 PRINT 9;9
30 PRINT 9,9
```

A linha 10 numera as colunas no topo da tela. Uma vírgula separa os números em "campos", cada um com dez colunas de extensão (dezesseis, no Sinclair). Agora tente o mesmo programa com textos (isto é, cordões alfanuméricos), acrescentando essas linhas:

```
40 LET A$="A"
50 PRINT A$;A$
60 PRINT A$,A$
```

Normalmente, costumamos alinhar palavras à esquerda e números à direita, mas os computadores descritos aqui não fazem isto automaticamente.

O ponto e vírgula é essencial para que a função **TAB** funcione corretamente, pois uma vírgula após um **TAB** provocará o deslocamento para uma nova posição de tabulação. O ponto e vírgula também é útil no final de uma declaração **PRINT**. Sua função, neste caso, é manter o cursor de impressão na tela, na última posição impressa, de tal forma que o próximo comando **PRINT** encontrado pelo programa comece a imprimir nessa posição, sem mudar de li-

nha na tela. O programa a seguir imprime todo o alfabeto utilizando um laço **FOR...NEXT** para incrementar de um em um os códigos ASCII para os caracteres de A a Z.

```
20 FOR C=65 TO 90
30 PRINT CHR$(C);
40 NEXT C
```

(Para micros da linha Sinclair ZX-81, substitua a linha 20 por **FOR C = 38 TO 63**, pois eles usam um código diferente do ASCII.) Se você não colocar o ponto e vírgula ao final da linha 30, cada letra será impressa em uma linha separada. Os sinais de pontuação e o **TAB** oferecem tantas maneiras de se exibir textos na tela que você encontrará certamente combinações adequadas aos seus próprios programas.

As linhas Sinclair contam com três declarações para posicionamento de textos na tela: **PRINT, PRINT AT** e **PRINT TAB**. A declaração **PRINT** sozinha imprime uma linha em branco (veja mais adiante).

A tela do Sinclair tem 22 linhas, numeradas de 0 a 21, a partir do topo da tela. Cada linha, por sua vez, tem 32 "colunas", ou posições, para os caracteres, numerados de 0 a 31, a partir da margem esquerda da tela. O comando **PRINT AT** indica onde o texto a ser impresso vai começar: os números (ou expressões) que seguem **AT** indicam a linha e a coluna; as linhas:

```
PRINT AT 0,0,"*"
PRINT AT 21,0,"*"
PRINT AT 0,31,"*"
PRINT AT 21,31,"*"
```

imprimirão um asterisco em cada canto do vídeo. Se você quiser imprimir mais de um caractere em uma locação da tela, o computador imprimirá o primeiro caractere onde você determinar e o restante nas demais posições da mesma linha.

```
PRINT AT 10,12;"COMPRIMENTO"
```

coloca o "C" na linha 10, coluna 12, e as outras letras em 10,13 , 10,14.... etc., até 10,17.

Outro recurso interessante do **PRINT AT** é que uma linha **PRINT** pode conter vários **AT**, separados por pontos e vírgulas. Por exemplo, a linha usada para imprimir quatro asteriscos poderia ser assim:

```
PRINT AT 0,0,"*"; AT 21,0,"*";
    AT 0,31,"*"; AT 21,31,"*"
```

PRINT TAB

No Sinclair, a instrução **PRINT TAB** funciona, em muitos casos, do mesmo modo que **PRINT AT**. Contudo, existem duas diferenças: não é preciso especificar a linha; e não se pode utilizar o **PRINT TAB** para escrever (nem para apagar) por cima de qualquer material na tela. Se você entrar um novo **PRINT TAB** na antiga posição, a impressão será realizada em uma linha abaixo da anterior. Agora, digite:

```
PRINT AT 0,15;"*"
PRINT AT 0,15;"?"
```

Limpe a tela (**CLS**) e tente isso:

```
PRINT TAB 15;"*";
PRINT TAB 15;"?";
```

Comparado com o **PRINT AT**, o **PRINT TAB** economiza trabalho de digitação e isenta o operador da tarefa de lembrar-se dos números de linhas à medida que vai prosseguindo.

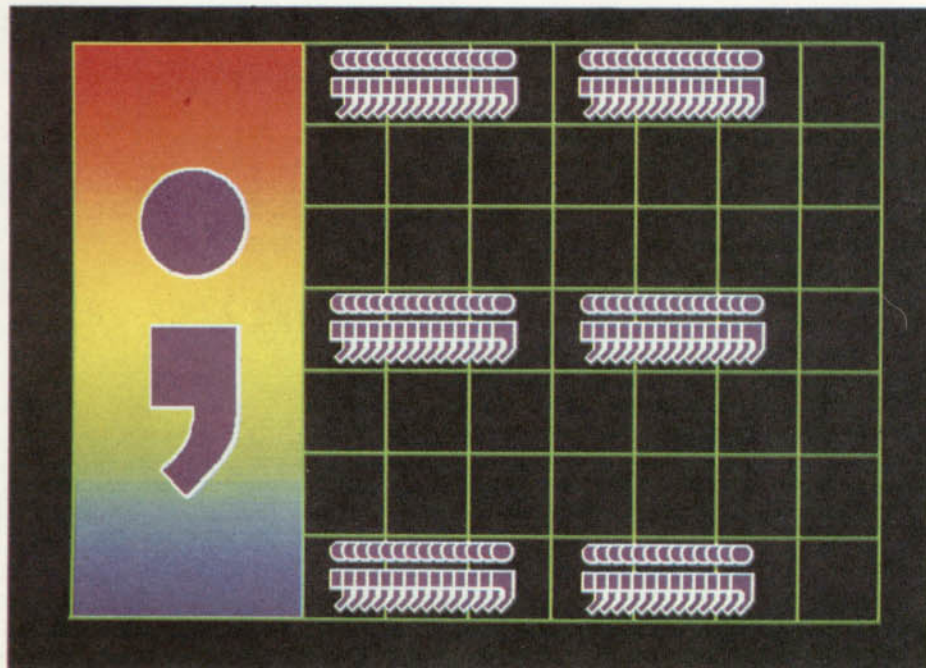
O programa a seguir permite que você digite as notas dos alunos de uma classe e as disponha ordenadamente na tela, usando uma série de funções **TAB**; além disso, ele calcula a média das notas (digite tudo em letras maiúsculas, para micros da linha ZX-81):

```
10 PRINT "NOME";TAB 12;"PROVA"
;TAB 18;"ORAL";TAB 23;"EXAME";
TAB 29;"MED"
20 PRINT AT 20,0;"Nome ?"
30 INPUT n$
35 PRINT AT 20,0;"Prova?"
40 INPUT np
45 PRINT AT 20,0;"Oral ?"
50 INPUT no
55 PRINT AT 20,0;"Exame?"
60 INPUT ne
70 LET m=((np+no*2)/3+ne)/2
80 PRINT N$;TAB 14;np; TAB 19;
no; TAB 24;ne; TAB 29; m
90 PRINT
100 PAUSE 200
110 GOTO 20
```

As linhas **PRINT**, usadas para pedir os dados iniciais de cada aluno, são posicionadas com o **AT** na penúltima linha da tela.

Ao ser rodado pela primeira vez, o programa limpa a tela e coloca na primeira linha as identificações correspondentes às colunas de dados. Em seguida ele perguntará: "NOME?". Digite o nome do aluno. Depois, perguntará sucessivamente as notas da prova, do exame oral e do exame final. Entre números inteiros ou, no máximo, com uma decimal. Em seguida o programa mostrará esses dados de entrada, juntamente com a média. Adicione as linhas:

```
75 LET M=INT(M*10)/10
76 LET N$=N$(TO 13)
```

O primeiro problema é que a média calculada pela última parte da linha 80 pode conter muitas casas decimais. Isto é resolvido pela linha 75, por meio de uma operação de *arredondamento* do resultado: multiplicamos a média (M) por 10, tomamos o valor inteiro do resultado e o dividimos por

Utilizado tanto pelo comando *PRINT* como pelo *INPUT*, o ponto e vírgula serve para controlar o modo pelo qual a informação é exibida na tela. Cada item é colocado no vídeo, seguindo imediatamente o item anterior na mesma linha.

10. Vejamos a média 7,654321: multiplicada por 10, ficaria 76,54321; seu inteiro é 76 que, dividido por 10, é 7,6.

Outro problema é que a primeira nota (**NP**) será mostrada na tela a partir da coluna 14. Isto quer dizer que o nome do aluno não deve ter mais do que treze caracteres. A função **TO** na linha 77 "amputa" o nome do aluno de modo a caber nesse espaço. Para interromper o programa sem limpar a tela, pressione <**CAPS SHIFT**> junto com <**BREAK**> (no ZX-81, pressione apenas <**BREAK**>). A linha 100 lhe dará tempo de fazer isso.

O programa abaixo oferece o esboço de uma contabilidade doméstica:

```
10 PRINT "NOME";TAB 10;"ITEM" ;
TAB 26;"PRECO";
15 LET total=0
20 FOR I=1 TO 8
35 PRINT AT 20,0;"Data ?"
40 INPUT d$
45 PRINT AT 20,0;"Item ?"
50 INPUT i$
55 PRINT AT 20,0;"Preco?"
60 INPUT p
70 LET total=total+p
80 PRINT d$;TAB 10;i$;TAB 26;p
90 PRINT
100 NEXT I
110 PRINT TAB 26;"------"
120 PRINT TAB 10;"TOTAL";TAB
26; total
```

Use corretamente a pontuação, com a declaração **PRINT TAB**. Normalmente, dois pontos e vírgulas são utilizados após **PRINT TAB**:

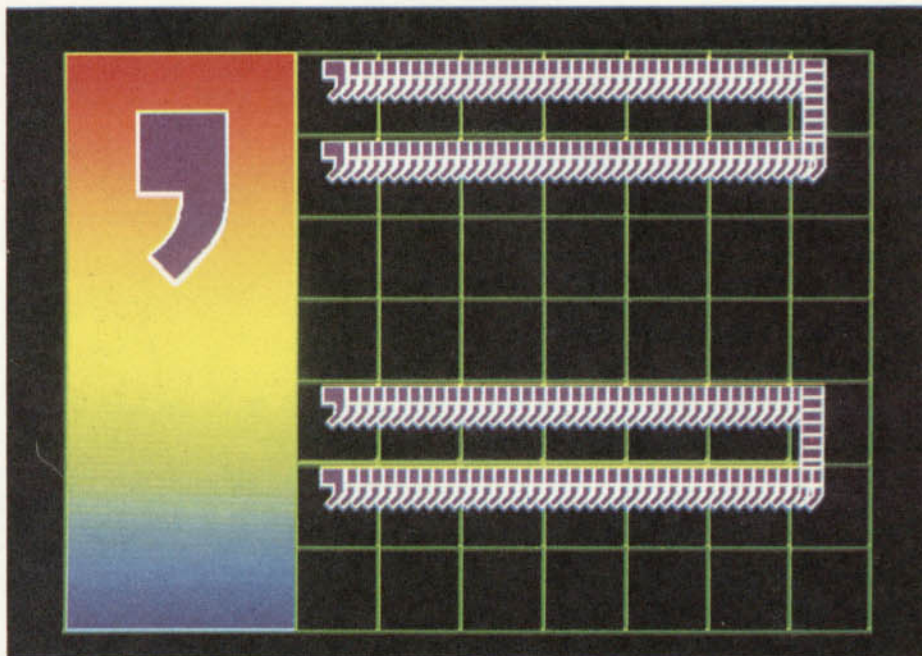
```
10 FOR n=1 TO 21
20 FOR t=0 TO 24 STEP 6
30 PRINT TAB t;n;
40 NEXT t
50 NEXT n
```

Quando tiver rodado o programa, modifique a linha 30 para:

```
O ponto e virgula e' usado para
mensagens uma apos a outra:

mensagemmensagemmensagemmensagem
mensagemmensagemmensagemmensagem
mensagemmensagemmensagemmensagem
mensagemmensagemmensagemmensagem
mensagemmensagemmensagemmensagem
mensagemmensagemmensagemmensagem
mensagemmensagemmensagemmensagem
mensagemmensagemmensagemmensagem
mensagemmensagemmensagemmensagem
mensagemmensagemmensagemmensagem
mensagemmensagemmensagemmensagem
mensagemmensagemmensagemmensagem
mensagemmensagemmensagemmensagem
mensagemmensagemmensagemmensagem
mensagemmensagemmensagemmensagem
mensagemmensagemmensagemmensagem
mensagemmensagemmensagemmensagem
```

Veja como usar um ponto e vírgula para preencher a tela com uma mensagem (exemplo rodado no Spectrum).

```
10 PRINT"mensagem"
20 GOTO 10
```

Os apóstrofos instruem o computador a colocar o item seguinte em uma nova linha, quando a máquina estiver impossibilitada de fazer isso automaticamente. Nem todos os computadores, entretanto, contam com esse recurso de pontuação.

```
30 PRINT TAB t,n;
```

... e então compare-a com essa:

```
30 PRINT TAB t;n
```

Um ponto e vírgula significa: "Aproxime bem a próxima palavra, sem nenhum espaço"; uma vírgula significa: "Comece a próxima palavra no início da coluna 0 ou da coluna 15"; uma declaração **PRINT TAB** precisa sempre de dois pontos e vírgulas (do contrário, a desordem se instalará na tela).

A maneira mais fácil de exibir uma mensagem na tela é:

```
PRINT "OLA"
```

Se PRINT "OLA" estiver na última linha da tela, esta rolará para cima, de modo a dar espaço à mensagem (não se esqueça de digitar as aspas). Limpe a tela antes e tudo ficará claro:

```
10 CLS
20 PRINT "OLA"
```

Agora "OLA" aparece no topo à esquerda. Acrescente essa linha e rode o programa:

```
30 PRINT "ADEUS"
```

"ADEUS" aparecerá abaixo de "OLA". Se quiser uma linha branca entre "OLA" e "ADEUS", acrescente:

```
25 PRINT
```

Agora você já sabe como suas mensagens são organizadas verticalmente na tela, mas ainda não sabe como elas aparecerão em cada linha. Suponha que você queira exibir "OLA" no meio da linha. O micro tem uma função (**TAB**) que coloca a mensagem no ponto da linha que você quiser. Acrescente essa linha e rode o programa:

```
40 PRINT TAB(15);"BOA TARDE!"
```

Os espaços em cada linha são numerados de 0 a 31 no TRS-Color, e de 0 a 63 no TRS-80. Então, a mensagem na linha 40 começará no espaço número 15. Não deixe espaço entre **TAB** e o parêntese, no TRS-Color; senão, (15) será tratado como variável e **TAB** não funcionará. Veja como **TAB** calcula o quadrado, o cubo, a raiz quadrada e o inverso dos números de 1 a 12:

```
10 CLS
20 PRINT "NUMERO";TAB(8);"CUBO"
;TAB(14);"QUAD.";TAB(21);"RAIZ"
;TAB(27);"INVER"
30 FOR J=1 TO 12
40 PRINT TAB(1);J;TAB(7);J*J*J;
TAB(13);J*J;TAB(20);INT(SQR(J)*
1000)/1000;TAB(26);INT(1000/J)/
1000
50 NEXT J
```

Um problema que deve ser resolvido neste programa, com um pouco de aritmética elementar, é que a raiz quadrada e o inverso do número J, calculados pela última parte da linha 40, podem ter

muitas casas decimais. Isso é resolvido por meio de uma operação de *arredondamento* do resultado para conter apenas uma casa decimal: multiplicamos a raiz quadrada por 1000, tomamos o valor inteiro do resultado, e o dividimos por 1000. Vejamos, por exemplo, a raiz 7,654321: multiplicada por 1000 ficaria 7654,321; seu inteiro é 7654 que, dividido por 1000, ficaria 7,654. Se você quiser imprimir em um lugar específico na tela deve utilizar **PRINT@** ("PRINT arroba") em vez de **PRINT TAB**.

```
10 CLS
20 PRINT @461,"BAIXO"
30 PRINT @77,"ALTO"
40 PRINT @257,"ESQUERDA"
50 PRINT @280,"DIREITA"
```

```
10 CLS
20 PRINT @925,"BAIXO"
30 PRINT @30,"ALTO"
40 PRINT @448,"ESQUERDA"
50 PRINT @502,"DIREITA"
```

Os números após **PRINT@** se referem às locações de tela (numeradas da esquerda para a direita, no TRS). O TRS-80 tem posições numeradas de 0 a 1023, e o TRS-Color (32 colunas) de 0 a 511. Eis um programa que calcula as notas dos alunos de uma classe:

```
5 CLS
10 PRINT "NOME";TAB(12);"PROVA"
;TAB(18);"ORAL"; TAB(23);
"EXAME";TAB(29);"MED"
20 PRINT @448,"NOME ";
30 INPUT N$
35 PRINT @448,"PROVA";
40 INPUT NP
45 PRINT @448,"ORAL ";
50 INPUT NO
55 PRINT @448,"EXAME";
60 INPUT NE
70 LET M=((NP+NO*2)/3+NE)/2
80 PRINT N$;TAB(14);NP;TAB(19);
NO; TAB(24);NE;TAB(29);M
90 PRINT
100 FOR I=1 TO 400:NEXT I
110 GOTO 20
```

```
5 CLS
10 PRINT "NOME";TAB(12);"PROVA"
;TAB(18);"ORAL"; TAB(23);
"EXAME";TAB(29);"MED"
20 PRINT @896,"NOME ";
30 INPUT N$
35 PRINT @896,"PROVA";
```

```
Colocando um apostrofe (no Spec-
trum, ou deixando de por o ponto
e virgula produz este efeito:

mensagem
mensagem
mensagem
mensagem
mensagem
mensagem
mensagem
mensagem
mensagem
mensagem
mensagem
mensagem
mensagem
mensagem
mensagem
mensagem
mensagem
```

Um outro exemplo de tela mostra a atuação do apóstrofo em uma linha de programa para o Spectrum:

```
10 PRINT"mensagem"
20 GOTO 10
```

```
40 INPUT NP
45 PRINT @896,"ORAL ";
50 INPUT NO
55 PRINT @896,"EXAME";
60 INPUT NE
70 LET M=((NP+NO*2)/3+NE)/2
80 PRINT N$;TAB(14);NP;TAB(19);
NO; TAB(24);NE;TAB(29);M
90 PRINT
100 FOR I=1 TO 400:NEXT I
110 GOTO 20
```

A linha 20 exibe os cabeçalhos das colunas. As linhas 40 a 70 limpam uma linha da tela e perguntam pelos dados necessários (**PRINT@** posiciona a linha que pede a informação na penúltima linha da tela). A linha 80 tabula as informações, e a 90 calcula a média. A variável **PA** faz a máquina checar se a tela foi preenchida.

PONTUAÇÃO

As vírgulas e os pontos e vírgulas controlam a posição do cursor. Agora digite:

```
10 CLS
20 PRINT"O CURSOR RETORNARA"
30 PRINT"SEM O PONTO-E-VIRGULA"
40 PRINT"MAS COM O PONTO-E-
VIRGULA ";
50 PRINT"ACONTECE ISTO"
```

Um ponto e vírgula ao final de uma linha **PRINT** obriga o cursor a permanecer onde estava quando terminou de imprimir. Acrescente essas linhas:

```
60 PRINT"E QUE TAL",
70 PRINT"USAR",
80 PRINT"VIRGULAS",
90 PRINT"?"
```

No TRS-Color: uma vírgula no fim de um **PRINT** faz o cursor saltar para a outra metade da tela. TRS-80: cada vírgula encontrada tabula para a próxima posição).

Os controles de cursor (declaração **LOCATE**), a função **TAB** e o comando **PRINT** com sinais de pontuação servem para melhorar a aparência das exibições na tela do MSX. A maneira mais fácil de mostrar uma mensagem é:

```
PRINT "BOM DIA!"
```

O "BOM DIA" aparecerá à esquerda da tela na próxima linha disponível. Se **PRINT "BOM DIA"** estiver na última linha da tela, esta rolará para cima, de modo a dar espaço à mensagem. Não se esqueça de digitar as aspas, ou obterá uma mensagem de erro. O resultado do comando acima ficará mais claro se você limpar a tela antes:

```
10 CLS
20 PRINT"BOM DIA!"
```

A FUNÇÃO TAB

De todas as instruções de posicionamento do cursor, a função **TAB** é, certamente, a mais utilizada. Ela se comporta, na tela, como o recurso de tabulação das máquinas de escrever, posicionando o cursor alguns espaços a partir da borda esquerda da área de texto da tela. Sua forma pode ser:

```
30 PRINTTAB(15)"BOA TARDE!"
```

Você pode colocar, se quiser, um ponto e vírgula depois do parêntese, mas isso não é necessário. Não deve existir espaço entre **TAB** e seu argumento entre parênteses, cujo valor está no ponto inicial do cordão impresso.

Funções **TAB** múltiplas podem ser utilizadas para posicionar várias observações em cada linha da tela.

```
30 PRINTTAB(10)"BOA" TAB(20)"TA
RDE!"
```

Neste caso, B é impresso na coluna 10, e T, na 20 (o valor do argumento sempre se refere à distância da borda esquerda da área de texto).

TAB é a abreviação de "tabular". Veja no programa a seguir como essa função pode ser útil (ela calcula o quadrado, o cubo, a raiz quadrada e o inverso dos números de 1 a 12):

```
10 CLS
20 PRINT"NÚMERO"TAB(10)"QUADRAD
O"TAB(20)"CUBO"TAB(30)"4º POT"
30 FORJ=1TO20
40 PRINTJ;TAB(10)J*J;TAB(20)J*J
*J;TAB(30)J*J*J*J
50 NEXT
```

Ao rodar o programa, você verá uma tabela de números sendo montada. Toda vez que o programa passar pelo laço, uma outra linha de números será exibida na tela (**TAB** permite colocar cada número na coluna correta).

POSICIONE O CURSOR

Suponhamos que você queira imprimir em um lugar específico na tela, e não apenas na próxima linha disponível. Neste caso, **PRINT TAB** não é o mais indicado: use **LOCATE**.

No MSX **LOCATE** pode ser usada para controlar o posicionamento do cursor em qualquer ponto da tela, como ocorre com os comandos **PRINT AT**, **PRINT@** e **HTAB/VTAB** de outras máquinas. Tente o programa abaixo:

```
10 CLS
15 LOCATE 14,23
20 PRINT"SUL"
25 LOCATE 12,1
30 PRINT"NORTE"
35 LOCATE 0,12
40 PRINT"OESTE"
45 LOCATE 25,12
50 PRINT"LESTE"
```

O programa escreve os pontos cardeais junto às quatro margens da tela.

A declaração **LOCATE** define, com dois números, a próxima posição do cursor da tela de textos. O primeiro argumento deve ser um número entre 0 e 39 e diz ao computador da coluna a ser posicionada. O segundo, um número de 0 a 24, indica a posição da linha a ser posicionada. Os argumentos que seguem **LOCATE** podem ser números, variáveis ou expressões com resultado numérico; e somente será considerado o seu valor inteiro (não fracionário) para o cálculo da posição.

Assim, para posicionar uma declaração **PRINT** em um ponto escolhido, estabeleça a posição X (para coluna) e Y (para carreira) e coloque no programa um **LOCATE** antes do **PRINT**.

COMO POSICIONAR ENTRADAS

As entradas de dados através do comando **INPUT** podem ser posicionadas da mesma forma que as mensagens que usam o comando **PRINT**:

```
10 CLS
20 LOCATE 10,10:INPUT"SEU NOME"
;N$
30 LOCATE 12,22:PRINT "OLA, "
;N$;"!"
```

Esse programa imprimirá a primeira instrução na coluna 10 e na linha 10, orientando o usuário com uma mensagem seguida de um ponto de interrogação, para efetuar uma entrada de dados com o **INPUT**. Qualquer coisa que seja digitada será armazenada na variável **N$**. A linha 30 apenas imprime o nome, desta vez na coluna 12 da linha 22. O **LOCATE** não precisa aparecer necessariamente na mesma linha que o **INPUT** e o **PRINT**. Eis aqui um programa que utiliza tanto **LOCATE** quanto **PRINT TAB** e que serve para calcular e exibir as notas dos alunos de uma classe:

```
10 CLS:KEYOFF
20 PRINT"NOME"TAB(10)"NOTA 1"TA
B(20)"NOTA 2"TAB(30)"MEDIA"
25 FORX=1TO15
30 LOCATE0,22:INPUT"NOME ";N$
40 LOCATE0,22:INPUT"NOTA 1";N1
50 LOCATE0,22:INPUT"NOTA 2";N2
60 LOCATE0,X:PRINTN$;TAB(10)N1;
TAB(20)N2;TAB(30)(N1+N2)/2
70 NEXT
```

A linha 20 exibe os cabeçalhos das colunas no topo da tela. As linhas 30 a 50 perguntam pelos dados necessários. **LOCATE** é necessário para posicionar a linha que pede a informação na penúltima linha da tela; sua não inclusão desorganizaria a tabela que está sendo montada na parte de cima. A linha 60 tabula a informação que você acabou de fornecer à máquina e calcula a média.

PONTUAÇÃO

O uso correto dos sinais de pontuação do **PRINT** é importante quando se está exibindo informações na tela. As vírgulas e os pontos e vírgulas controlam a posição do cursor (o quadrado ou o sinal de sublinha que assinala onde aparecem os caracteres).

```
10 CLS
20 PRINT"O CURSOS MUDA DE LINHA
"
30 PRINT"SEM O PONTO-E-VÍRGULA"
40 PRINT"MAS, COM ELE ";
50 PRINT"TUDO MUDA!"
```

Quando se inclui um ponto e vírgula ao final de uma linha **PRINT** no programa, o cursor não retorna ao início da próxima linha: ele permanece exatamente onde estava quando terminou de imprimir. A próxima declaração **PRINT** continua diretamente depois da última declaração. Acrescente essas linhas ao programa e rode-o:

```
60 PRINT
70 PRINT"E QUE TAL",
80 PRINT"USAR VÍRGULAS?",
90 PRINT"E VER",
100 PRINT"O QUE ACONTECE?"
```

```
A vírgula imprime a mensagem em
colunas, deste jeito:

mensagem            mensagem
mensagem            mensagem
mensagem            mensagem
mensagem            mensagem
mensagem            mensagem
mensagem            mensagem
mensagem            mensagem
mensagem            mensagem
mensagem            mensagem
mensagem            mensagem
mensagem            mensagem
mensagem            mensagem
mensagem            mensagem
mensagem            mensagem
mensagem            mensagem
mensagem            mensagem
```

Neste exemplo, igualmente produzido a partir da tela do Spectrum, a vírgula imprimirá em colunas. Assim:

```
10 PRINT "mensagem"
20 GOTO 10
```

Os computadores das linhas TRS-Color, MSX, TRS-80 e Apple II contam com um maior número de colunas disponíveis.

Você provavelmente já conhece este comando:

```
PRINT "BOM DIA!"
```

O seu computador imprimirá a mensagem na primeira linha disponível, começando no limite esquerdo da tela. Se você digitar o programa:

```
10   HOME
20   PRINT "BOM DIA!"
```

verá que a tela será apagada e que a sua mensagem aparecerá na primeira linha, como se tivesse dado uma folha de papel em branco ao computador.

Mas é possível fazer com que a mensagem apareça numa posição diferente. Adicione essa linha:

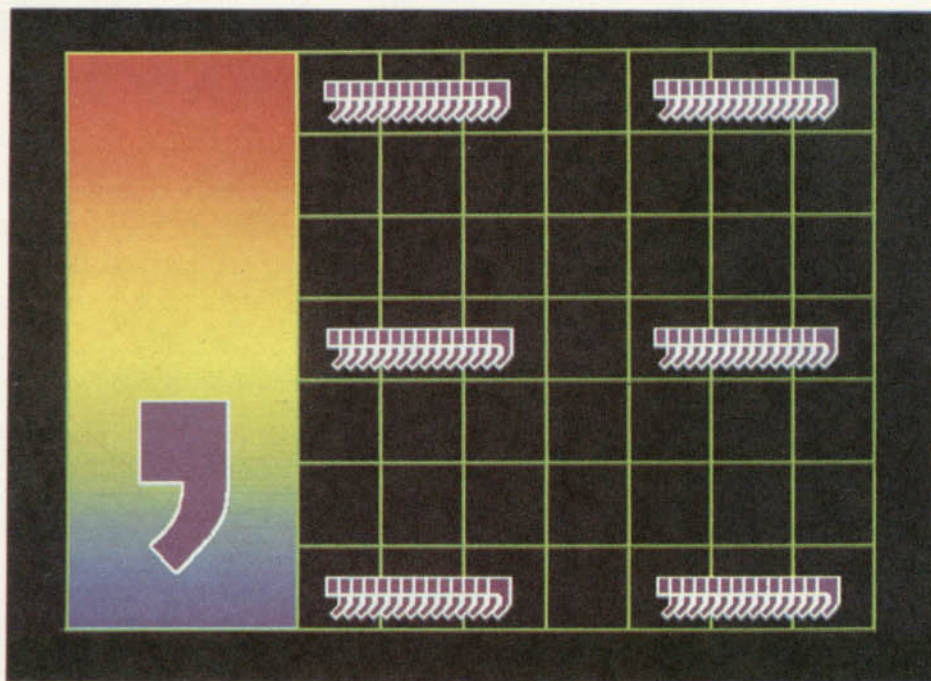
```
30   PRINT  TAB( 15)"BOA TARDE!"
```

A mensagem aparece centralizada na tela. A primeira letra foi colocada na coluna 15 da linha de impressão. A tela de textos do Apple II tem 24 linhas de quarenta colunas de largura, cada. Você também pode usar mais que um **TAB** por linha:

```
40   PRINT  TAB( 10)"DIA"; TAB(
20)"TARDE"; TAB( 30)"NOITE"
```

E se você quiser pular linhas? É fácil: experimente adicionar as linhas abaixo ao seu programa:

```
25   PRINT
35   PRINT
```

Um comando **PRINT** sem nada depois significa simplesmente uma linha em branco na tela.

Nas declarações *PRINT* e *INPUT*, as vírgulas indicam que a informação será exibida em campos. O ZX-81, o Spectrum e o TRS-Color têm dois campos de dezesseis colunas; o MSX e o Apple II, três campos de oito colunas; já o TRS-80 conta com quatro campos.

HTAB E VTAB

O Apple II tem outro sistema para posicionar mensagens na tela.

```
10 HOME
30 HTAB 15 : PRINT "BOA TARDE"
```

Observe que o efeito é semelhante ao do **PRINT TAB**. Acrescente a linha a seguir e rode o programa novamente:

```
20 VTAB 15
```

A sua mensagem está agora no centro da tela. Com esses dois comandos, **HTAB** (abreviatura de *horizontal tabulation*) e **VTAB** (*vertical tabulation*), você pode colocar o cursor (e suas mensagens) em qualquer lugar da tela.

TABULE AS ENTRADAS

As entradas de dados através do **INPUT** podem ser posicionadas da mesma forma que as mensagens com **PRINT**:

```
10  HOME
20  VTAB 10: HTAB 10: INPUT "SE
U NOME? ";N$
30  VTAB 22: HTAB 12: PRINT "OL
A, ";N$;"!"
```

A primeira instrução será impressa na coluna 10 e na linha 10, orientando o usuário com uma mensagem, seguida de um ponto de interrogação, para entrar dados com o **INPUT**. Qualquer coisa que seja digitada será armazenada na variável **N$**. A linha 30 imprime o nome, desta vez na coluna 12 da linha 22. A ordem em que **VTAB** e **HTAB** aparecem no programa não é importante. Se você não quiser um ponto de interrogação, omita-o da linha 20.

```
INPUT"ENTRE O CUSTO EM CZ$:";
CUSTO
```

Um ponto de interrogação prejudicaria a clareza da tela. O programa abaixo serve para calcular e exibir as notas dos alunos de uma classe:

```
10  HOME
20  INPUT "QUANTOS ALUNOS? ";A
30  HOME : PRINT "NOTA1","NOTA2
","MEDIA"
40  FOR J = 1 TO A
50  VTAB 21: INPUT "NOTA1? ";N1

60  VTAB 21: INPUT "NOTA2? ";N2

70  VTAB J + 3: PRINT N1,N2,(N1
 + N2) / 2
80  NEXT
```

A linha 20 exibe os cabeçalhos das colunas no topo da tela. As linhas 40 a 70 limpam uma linha e perguntam pelos dados necessários. O **HTAB** e o **VTAB** antes do **INPUT** são necessários para posicionar a linha que pede a informação na penúltima linha da tela.

A linha 80 tabula a informação que você acabou de fornecer à máquina, e a linha 90 calcula a média.

A média das notas pode dar um resultado com muitas casas decimais (uma dízima periódica, por exemplo). Ora, esse resultado não caberia na linha de tela do Apple II, extravasando para a linha seguinte e ''estragando'' o aspecto da tela. Isso pode ser resolvido por meio de uma operação de arredondamento para que o resultado contenha apenas uma casa decimal: multiplicamos a raiz quadrada por 1000, tomamos o valor inteiro do resultado, e o dividimos por 1000. Vejamos, por exemplo, a média 7,654321: multiplicada por 1000 ficaria 7654,321; seu inteiro é 7654 que dividido por 1000 ficaria 7,654. Se você quiser um resultado com duas casas decimais, ao invés de três, multiplique e divida por 100. Substitua a linha:

```
70 VTAB J + 3: PRINT N1,N2,
INT((N1+N2)/2/10)*10
```

PONTUAÇÃO

Em BASIC, a pontuação oferece três possibilidades: a vírgula, o ponto e vírgula e a ausência de pontuação. O ponto e vírgula faz com que o cursor não mude de posição após imprimir uma mensagem na tela, o que resulta em uma justaposição, se várias delas forem impressas em seqüência:

```
10  HOME
20 A = 10:B = 20:C = 30
30  PRINT A;B;C
```

A vírgula produz um efeito bem diferente. Tente adicionar essa linha e veja o que acontece:

```
40  PRINT A,B,C
```

A tela, neste caso, é dividida em três colunas de tabulação automática e tudo que for mostrado na tela obedecerá a essa tabulação, exceto se algum dado exceder o tamanho da coluna.

Neste programa mostramos como a tabulação automática com a vírgula pode ser útil (ele calcula o quadrado e o cubo dos números de 1 a 20):

```
10  HOME
20  PRINT "NUMERO","QUADR","CUB
O"
25  PRINT
30  FOR J = 1 TO 20
40  PRINT J,J * J,J * J * J
50  NEXT
```

Você verá uma tabela de números sendo montada. Toda vez que o programa passar pelo laço uma outra linha de números será exibida na tela. A utilização do **TAB** permitirá que você coloque cada número na coluna correta.

TORNE O JOGO MAIS DIFÍCIL

- COMO DESENHAR UM LABIRINTO ALEATÓRIO
- DUAS MANEIRAS DE TORNAR O JOGO MAIS DIFÍCIL
- DESCUBRA O TESOURO

Se você gosta de enfrentar desafios, aprenda a tornar mais complicados seus jogos de labirinto, incorporando a eles níveis crescentes de dificuldade. E veja quanto tempo leva para encontrar um tesouro oculto no esconderijo de um sangüinário pirata.

Muitos programas pedem que se escolha um nível de dificuldade, antes de começar o jogo. Essa exigência permite que tanto os iniciantes quanto os especialistas possam utilizar o mesmo jogo, sem que este se torne excessivamente fácil ou difícil. Existem muitas maneiras de se introduzir níveis de dificuldade em um jogo, dependendo da sua natureza. Por exemplo: podemos mudar o número de inimigos enfrentados pelo jogador; desenvolver a ação em diferentes velocidades; conceder maior ou menor tempo para o jogo; variar os problemas a serem encontrados pelo jogador, e assim por diante.

Desta vez, vamos aprender a incorporar níveis de dificuldade a um programa de jogo de labirinto. O jogo escolhido emprega uma entre duas maneiras possíveis de se produzir níveis de dificuldade — o método a ser utilizado em detalhe dependerá do modelo de seu computador. O jogo não envolve apenas a tentativa de se achar o caminho através do labirinto, mas também uma limitação no tempo de que dispõe o jogador para guiar um homenzinho até um tesouro desenhado em algum lugar do labirinto. Não apresentaremos aqui uma versão do programa para microcomputadores da linha ZX-81, pois o jogo ficaria excessivamente lento. Também não será apresentada versão para micros da linha TRS-80, devido à baixa resolução gráfica do vídeo desses modelos.

Existem basicamente duas formas de aumentar a dificuldade de resolução de um labirinto: ou se altera o limite de

tempo, ou se varia a complexidade do labirinto. A razão pela qual devemos escolher métodos diversos, de acordo com o tipo do computador, tem a ver com a maneira como o labirinto é produzido. Esse exemplo mostra que você precisa decidir o caminho a seguir, quando quiser inventar jogos com diferentes níveis de dificuldade.

AS VIDAS

Uma das maneiras de tornar mais excitante um jogo de labirinto consiste em impor algum tipo de penalidade ao jogador que esgota seu tempo sem encontrar o tesouro. Pode-se, por exemplo, fazer com que ele perca alguns pontos na contagem; mas a penalidade mais comum é obrigá-lo a perder uma "vida".

Nesse jogo daremos três "vidas" ao jogador: se ele não achar o tesouro dentro do limite de tempo, em três tentativas, o jogo terminará.

LABIRINTOS ALEATÓRIOS

O jogo é baseado em uma sub-rotina de produção de labirintos aleatórios (ou seja, de desenho gerado ao acaso). É um programa muito interessante, pois desenha um labirinto totalmente diferente de cada vez, evitando assim que você tenha de projetar uma série enorme desses desenhos. Já mostramos, em um artigo anterior, como projetar labirintos, utilizando linhas **DATA** para incorporar o seu desenho ao programa. Imagine, então, como seria complicado incluir vários labirintos ao programa, usando essa técnica.

A geração aleatória de labirintos é muito mais fácil do què isto, embora apresente algumas dificuldades complementares. Uma maneira óbvia de projetá-los, por exemplo, seria desenhar blocos gráficos aleatoriamente na tela. O problema, neste caso, é que nada garante que se formem caminhos interiores entre os blocos; ou seja, pode não surgir um verdadeiro labirinto. Assim, caso escolhesse esse método, você teria que encontrar maneiras de checar a existência de, no mínimo, uma saída.

LABIRINTOS E CAMINHOS AO ACASO

O melhor modo de se desenhar labirintos aleatórios é inventar um programa que trace caminhos ao acaso, incluindo-os depois no desenho final. O programa proposto a seguir foi desenvolvido de tal forma que a linha do caminho aleatório fica contida no interior da moldura desenhada na tela. Não é permitido que a linha cruze com ela mesma. Quando o caminho aleatório não puder prosseguir, o programa fará o mesmo percurso de volta. Ele faz isso passo a passo, examinando a área em torno do caminho anteriormente traçado, até encontrar espaço livre para prosseguir. Quando esse espaço é encontrado, o programa inicia um outro desvio no caminho aleatório, prosseguindo por ele até ser imobilizado novamente, e assim por diante. O computador continua tentando desenhar novos caminhos até que toda a tela seja preenchida; nesse momento, ele volta ao ponto de partida.

Quando o programa acaba de desenhar o labirinto, apenas um caminho livre aparece na tela; esse caminho pode ser facilmente distinguido, pois os desvios não são muito complicados. O labirinto também será solucionável através da *regra da mão direita*. De acordo com essa regra, é sempre possível sair de um labirinto, seguindo-se a sua parede direita (ou esquerda, tanto faz, desde que seja sempre a mesma). Para impedir que alguém aplique a regra, basta construir algumas *ilhas* no labirinto, obrigando o pobre jogador a ficar dando voltas sem fim. Assim, após desenhar um labirinto, o programa traça um certo número de blocos aleatórios que fazem com que o desenho pareça mais complexo, impedindo que alguém recorra à regra da mão direita.

Uma vez digitado, guarde o programa em fita ou disco, pois o próximo artigo mostrará como acrescentar a ele alguns efeitos sonoros.

S

O programa para os micros da linha Spectrum começa estabelecendo os blocos gráficos, introduzindo as variáveis e fazendo uma preparação geral para o jogo. Digite essa seção do programa, mas não a execute ainda:

```
10 FOR n=0 TO 23: READ a:
POKE USR "a"+n,a: NEXT n
20 LET hs=0
30 INPUT "Selecione nivel (1
a 6) ";ta
40 LET ta=1100-100*ta
50 BORDER 1: PAPER 1: INK 0:
CLS : INK 7
60 LET s=0: LET vidas=3
70 PRINT  BRIGHT 1; PAPER 6;
INK 2;" SCORE          RECORDE
      "
490 DATA 24,24,60,82,82,24,36,
36,127,65,93,85,81,95,64,127,
24,24,255,255,24,24,24,24
```

A linha 10 define os blocos gráficos para o jogo — um homenzinho, o tesouro e uma cruz — por meio da leitura dos dados na linha 490. A variável que armazenará o recorde — **HS** (*high score*) — é zerada na linha 20.

A seguir, pede-se ao jogador para escolher um nível de dificuldade de 1 a 6 (**TA**). Quanto menor for o número, mais baixo será esse nível (e, portanto, mais fácil será o jogo). Você verá que na linha 40 os números mais baixos estabelecem tempos maiores, e os números mais altos, tempo menores.

As cores da tela e dos gráficos são estabelecidas na linha 50. A linha 60 zera o placar e atribui as três "vidas" ao jogador. A linha 70, finalmente, exibe as palavras ESCORE (contagem) e RECORDE, juntamente com os espaços em branco para colocar os números correspondentes, na tela.

DESENHE O LABIRINTO

Agora, digite essas linhas:

```
80 FOR n=22561 TO 22589: POKE
n,16:   POKE n+640,16: NEXT n
90 FOR n=1 TO 21: POKE 22528+
n*32,16  : POKE 22558+n*32,16
: POKE 22559+n*32,9: NEXT n
100 LET b=22593: LET a=b
110 DIM a(4): LET a(1)=-1: LET
a(2)=- 32: LET a(3)=1: LET a(4
)=32
120 POKE a,56
130 LET j=INT (RND*4)+1: LET g
=j
140 LET b=a+a(j)*2: IF PEEK b=
8 THEN  POKE b,j: POKE a+a(j),
56: LET a=b : GOTO 130
150 LET j=j+1: IF j=5 THEN
LET j=1
160 IF j<>g THEN  GOTO 140
170 LET j=PEEK a: POKE a,56:
IF j<5   THEN  LET a=a-a(j)*2:
GOTO 130
180 POKE 22625,56
190 FOR n=1 TO 20
200 LET k=22528+64*(INT (RND*9
)+2)+   INT (RND*29)+1
210 POKE k,56: NEXT n
```

A borda do labirinto é traçada pelas linhas 80 e 90, colocando-se o código 16 por meio de comandos **POKE** nas áreas de memória de vídeo. Esse código define a cor de fundo (**PAPER**), gerando portanto uma borda constituída de blocos vermelhos. As linhas 100 a 180 desenham o labirinto. Não caia na tentação de interromper o programa e limpar a tela nesse ponto. Se você fizer isso, o labirinto se perderá, pois ainda está armazenado apenas no arquivo de atributos. Para completar o labirinto, as linhas 190 a 210 exibem vinte quadrados em posições aleatórias dentro do labi-

rinto. Se um quadrado "cair" em cima de uma parede, uma passagem será automaticamente aberta.

COMO CRIAR UM JOGO

A próxima seção do programa diz respeito ao jogo em si (não tente ainda rodar o programa):

```
220 LET x=15: LET y=10
230 LET tx=INT (RND*15)*2+1
240 LET ty=INT (RND*10)*2+2
250 PRINT  BRIGHT 1; PAPER 2;
AT 0,7;s;AT 0,24;hs
260 POKE 23672,0: POKE 23673,0
270 PRINT  FLASH 1; PAPER 3;
INK 6;AT ty,tx;CHR$ 145
280 PRINT  INK 2; PAPER 7;AT y
,x;CHR$ 144
290 IF PEEK 23672+256*PEEK
23673>ta THEN  GOTO 390
300 IF INKEY$="" THEN  GOTO
290
310 LET a$=INKEY$: LET sx=x:
LET sy=y
320 IF a$="z" AND ATTR (y,x-1)
>=56 THEN  LET x=x-1
330 IF a$="x" AND ATTR (y,x+1)
>=56 THEN  LET x=x+1
340 IF a$="k" AND ATTR (y-1,x)
>=56 THEN  LET y=y-1
350 IF a$="m" AND ATTR (y+1,x)
>=56 THEN  LET y=y+1
360 PRINT  PAPER 7; INK 2;AT s
y,sx;" ";AT y,x;CHR$ 144
370 IF ty=y AND tx=x THEN
GOTO 470
380 GOTO 290
```

A posição inicial do homenzinho é estabelecida pela linha 220; o homenzinho começa sempre nas posições 15,10 no início de cada jogo.

O tesouro é colocado aleatoriamente pelas linhas 230 e 240, e a linha 270 o exibe na tela. A gama de escolha dos números aleatórios garante que o tesouro caia sempre em um caminho.

A linha 250 exibe os valores do escore e do recorde (ambos inicialmente em 0), ao passo que a linha 260 zera o cronômetro interno por meio de dois **POKE** em duas locações de memória. A linha 290 verifica se o limite de tempo foi excedido e, se isso aconteceu, pula para a linha 390.

O homenzinho é exibido pela linha 280. Note que os comandos **CHR$ 144** e **CHR$ 145** das linhas 270 e 280 são utilizados para exibir os blocos gráficos predefinidos para esses códigos. O **CHR$** é empregado aqui porque é mais fácil de ser visto na listagem do programa, embora você possa utilizar o método alternativo através de letras, como foi explicado anteriormente. As linhas restantes (ou seja, da 300 à 380) se ocupam com a movimentação do homenzinho no labirinto. As linhas 320 a 350 verificam se está sendo pressionada a tecla adequada, e se o próximo quadrado para a movimentação é um caminho e não uma parede. O teste utiliza o **ATTR**, que já foi explicado anteriormente. A linha 260 anula a última posição do homenzinho e o exibe novamente em uma posição diferente. A linha 370 testa se o homenzinho atingiu o tesouro. Se ele atingiu, a contagem de pontos é aumentada, antes de se sortear e exibir um outro tesouro a ser encontrado.

A MORTE DO PIRATA

A última seção final do programa é apresentada a seguir. Agora, finalmente, você pode rodá-lo.

```
390 PRINT  FLASH 1; PAPER 0;
INK 5;AT y,x;CHR$ 146
400 LET vidas=vidas-1: FOR f=1
TO 200: NEXT f: IF vidas>0
THEN  GOTO 260
410 IF s>hs THEN  LET hs=s
```

```
420 PRINT  BRIGHT 1; PAPER 2;
AT 0,24;hs
430 PRINT  FLASH 1;AT 10,1;" P
ressione qualquer tecla para
jogar de novo "
440 IF INKEY$<>"" THEN  GOTO
440
450 IF INKEY$="" THEN  GOTO
450
460 GOTO 30
470 LET s=s+ta-PEEK 23672+256*
PEEK 23673: GOTO 230
```

Quando o jogador perde uma "vida", e antes mesmo que a linha 400 subtraia 1 ao total de "vidas", uma cruz piscante (**CHR$ 146**) é exibida pela linha 390. Existe ainda uma pausa antes que o jogo retorne à linha 260, a qual coloca novamente o relógio em 0, de modo a ficar pronto para outra tentativa de atingir o tesouro. Esse retorno, evidentemente, só ocorrerá se o jogador ainda tiver mais "vidas" para usar.

Se não restar nenhuma "vida", a linha 410 compara o placar final com o último recorde. O recorde registrado é alterado quando o escore for maior. Em ambos os casos, a linha 420 exibe o recorde obtido até o momento.

A linha 410 é necessária para bloquear o jogo, caso o jogador ainda esteja pressionando uma das teclas de movimentação. A linha 420, por sua vez, bloqueia o programa até que uma nova pressão em qualquer tecla assinale que o jogador quer prosseguir.

A linha 470 calcula o escore, logo após o encontro do homem com o tesouro. O teste está na linha 370.

T

A tela de texto proporciona a maneira mais fácil de se desenhar um labirinto nos micros da linha TRS-Color. Mas os labirintos assim obtidos seriam muito simplificados por causa do grande tamanho e do pequeno número de blocos gráficos disponíveis.

Em vez disso, o programa aqui proposto desenha os labirintos na tela com gráficos de alta resolução. Embora seja mais complicado do que o que se pode conseguir na tela de textos, ele tem a vantagem de desenhar labirintos de diferentes complexidades, fornecendo assim vários níveis de dificuldade.

Imagine que o caminho seja formado por uma série de blocos quadrados. Se você quisesse desenhar um labirinto bem simples, bastaria escolher um bloco grande, ao passo que, se quisesse um labirinto mais complexo, deveria optar por um bloco de menor tamanho.

A primeira seção do programa introduz as variáveis e prepara o computador para desenhar labirintos aleatórios. Digite o programa, mas não execute ainda, senão você obterá um erro do tipo UL — linha indefinida —, quando o programa tentar executar o **GOSUB 1000** da linha 110.

```
10 PMODE 4,1
20 CLS:PRINT @193,"NIVEL DE DIF
ICULDADE (0-5)";
30 L$=INKEY$:IF L$<"0" OR L$>"5
" THEN 30
40 BS=12-VAL(L$):NX=2*INT(.5+12
8/BS):NY=2*INT(.5+96/BS)
50 SX=250-BS*NX:SY=190-BS*NY
60 DIMP(NX,NY),A(5),B(5)
70 PCLS 5: DRAW"S"+STR$(INT(8.5
-4*VAL(L$)/5))+"C0BM0,0BR2BDNFN
GD3NFG"
80 GET (0,0)-(BS-1,BS-1),A,G
90 GET(10,10)-(BS+9,BS+9),B,G:C
OLOR 5,0
100 CLS:PRINT @226,"GERANDO LAB
IRINTO DE NIVEL ";L$
110 GOSUB 1000
120 GOTO 120
```

A linha 10 diz ao computador para utilizar o quarto modo gráfico (**PMODE 4**) para todo o programa. A tela de alta resolução não é ligada nesse estágio. A linha 20 exibe, então, a mensagem NÍVEL DE DIFICULDADE (0 - 5).

L$** é o nível que o jogador escolheu. O valor numérico de **L$ — VAL (L$) na linha 40 — regula o tamanho do bloco, a extensão do caminho e a complexidade do labirinto. O **INKEY$** na linha 30 significa que o jogador não precisa digitar mais que o único dígito em resposta à pergunta, e que o programa continua sem que ele precise pressionar **<ENTER>**.

Na linha 40, o **BS** é o tamanho do bloco gráfico, em pixels. Esse tamanho pode variar de sete a doze pixels. **NX** é o número de blocos na direção horizontal, e **NY** é o número de blocos na direção vertical.

Antes de desenhar o labirinto na tela, o computador calculará sua aparência final e colocará essa informação no conjunto P, o qual é dimensionado (**DIM**) na linha 60. O conjunto A contém o formato do homenzinho, e o conjunto B, um espaço em branco, de modo a provocar sua animação gráfica. O homenzinho é desenhado pela linha 70. Nesse programa o homenzinho será desenhado em tamanho maior, quando o caminho do labirinto for mais largo, e menor quando for mais estreito.

Agora que o homenzinho foi desenhado, a linha 80 o coloca no conjunto A por intermédio de um comando **GET**, e a linha 90 preenche o conjunto B de branco. Nas lições anteriores, quando um espaço em branco era utilizado em um jogo, não era necessário colocar nada no conjunto: tínhamos apenas um conjunto vazio. Desta vez, é necessário um espaço em branco, de modo a com-

binar com a cor de fundo do caminho.

O comando **COLOR** na linha 90 assegura que o labirinto será desenhado mais tarde na cor correta (de outro modo você estaria desenhando um labirinto preto sobre um fundo preto).

A linha 100 diz ao jogador que o labirinto está sendo produzido; pode demorar algum tempo até o desenho aparecer.

DESENHE O LABIRINTO

Agora digite a sub-rotina de produção do labirinto — chamada pela linha 110 — e você poderá rodar o programa:

```
1000 FOR J=0 TO NX:P(J,NY)=6:P(
J,0)=6:NEXT
1010 FOR J=0 TO NY-2:P(0,J)=6:P
(NX,J)=6:NEXT
1020 X=2:Y=2:LX=2:LY=2
1030 J=RND(4)-1:G=J
1040 Y=LY+2*((J=0)-(J=2)):X=LX+
2*((J=3)-(J=1))
1050 IF P(X,Y)=0 THEN P(X,Y)=J+
1:P((X+LX)/2,(Y+LY)/2)=5:LX=X:L
Y=Y:GOTO 1030
1060 J=(J+1)AND 3: IF J<>G THEN
 1040
1070 J=P(LX,LY)-1:P(LX,LY)=5:IF
 J<4 THEN LX=LX-2*((J=3)-(J=1))
:LY=LY-2*((J=0)-(J=2)):GOTO 103
0
1080 FOR J=0 TO 20:P(2+2*RND((N
X-3)/2),1+RND(NY-3))=5:P(1+RND(
NX-3),2+2*RND((NY-3)/2))=5:NEXT
1090 SCREEN 1,1:PCLS
1100 FOR J=2 TO NX-2:FOR K=2 TO
 NY-2
1110 IF P(J,K)=5 THEN LINE(J*BS
+SX,K*BS+SY)-((J+1)*BS+SX-1,(K+
1)*BS+SY-1),PSET,BF
1120 NEXT K,J:RETURN
```

As linhas 1000 a 1080 "desenham" o labirinto na memória do computador e armazenam a sua forma no conjunto P — cada elemento do conjunto corresponde a um bloco do labirinto. A sub-rotina armazena o número 5 em P onde quer que exista um caminho, e o número 0, se existir uma parede. Uma vez que o labirinto tenha sido armazenado em P, a linha 1090 liga a tela de alta resolução e a deixa pronta para a execução do desenho.

As linhas 1100 e 1120 exibem o labirinto na tela mediante a verificação dos conteúdos de P. Quando um 5 for encontrado, um quadrado branco será impresso.

MOVIMENTO, TESOURO, VIDAS

Agora você precisa de um jogo para acompanhar o seu labirinto aleatório. Digite a próxima seção do programa, mas não o rode porque ele chamará uma sub-rotina que ainda não existe.

```
120 X=2:Y=2:LX=2:LY=2:TI=800:LI
=3
130 TIMER=0
140 X1=1+RND(NX-3):Y1=1+RND(NY-
3):IF P(X1,Y1)=5 THEN P(X1,Y1)=
7:DRAW "S4C0BM"+STR$(SX+X1*BS)+
","+STR$(SY+Y1*BS)+"BFR5D5L3U3R
D" ELSE 140
150 X1=X*BS+SX:Y1=Y*BS+SY
160 PUT(X1,Y1)-(X1+BS-1,Y1+BS-1
),A,PSET
170 IF PEEK (338)=251 THEN Y=Y-
1
180 IF PEEK (342)=253 THEN Y=Y+
1
190 IF PEEK (340)=247 THEN X=X-
1
200 IF PEEK (338)=247 THEN X=X+
1
210 IF P(X,Y)=7 THEN F=1:P(X,Y)
=5: GOTO 230
220 IF P(X,Y)<>5 THEN X=LX:Y=LY
:GOTO 170
230 IF X<>LX OR Y<>LY THEN PUT
(X1,Y1)-(X1+BS-1,Y1+BS-1),B,PSE
T:LX=X:LY=Y:FOR P=1 TO 68:NEXT
240 IF F=1 THEN F=0:SC=SC+(TI-T
IMER):TI=TI-10:GOTO 130
250 IF TIMER >TI THEN GOSUB 500
: IF LI<1 THEN 100
260 GOTO 150
```

A linha 120, que substitui a linha 120 da seção anterior, contém as variáveis a partir das quais é calculada a posição do homenzinho. X, Y é a localização atual do homenzinho, e LX e LY, sua última posição; porém, em virtude da largura variável do caminho, os valores devem ser ligeiramente ajustados um pouco antes de o homenzinho aparecer na tela. **TI** é o limite de tempo para descobrir o tesouro. Esse tempo limite é de dezesseis segundos. Se o homenzinho não tiver encontrado o tesouro, quando o tempo se esgotar, perderá uma "vida". No início do jogo, o jogador tem três "vidas" — **LI=3.**

O cronômetro interno do computador é zerado na linha 130, antes que a linha 140 selecione uma posição aleatória para o tesouro. O elemento correspondente em P é examinado para que haja certeza de que nesse lugar há um caminho e não uma parede. Se o tesouro estiver em um caminho, o valor do conjunto será modificado de 5 para 7. A última parte da linha desenha o tesouro no labirinto.

A posição do homenzinho é calculada na linha 150, levando-se em consideração a largura do caminho, que é o tamanho do bloco BS. A linha 160 coloca o homenzinho em posição.

As linhas 170 a 220 examinam o teclado por meio de comandos **PEEK,** de modo a permitir a movimentação do pirata no labirinto. Como não é possível deixá-lo atravessar paredes, a linha 220 o mantém dentro do caminho. A linha 210 verifica se o tesouro foi encontrado, examinando o elemento correspondente em P para o número 7. Se o computador o achar, o "indicador de descoberta" (F) será igualado a 1.

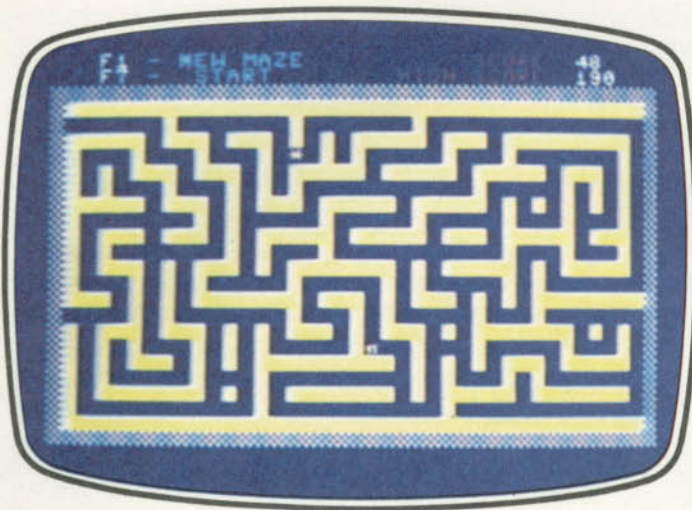
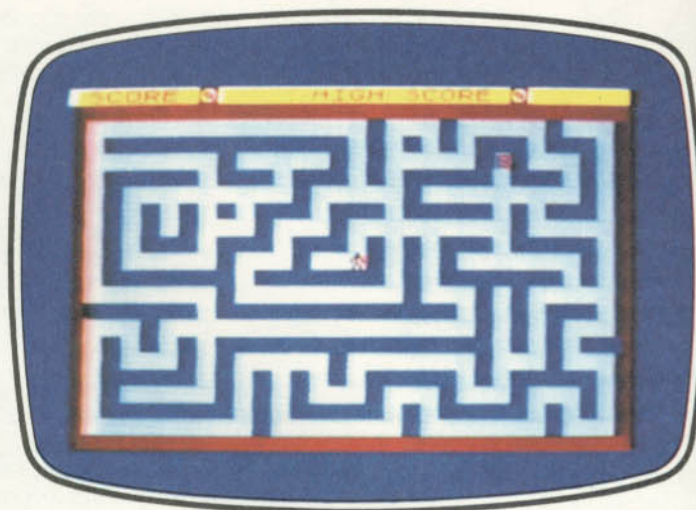
A linha 230 movimenta o pirata. O espaço em branco é colocado, com o comando **PUT,** sobre a última posição do personagem, e sua localização atual torna-se a última posição.

A linha 240 calcula o escore, caso o tesouro tenha sido encontrado. O limite de tempo é diminuído então de 10 segundos. O F volta ao 0, e o programa retorna para zerar o cronômetro. O programa continua, redesenhando o tesouro em outro lugar, deixando o pirata no mesmo ponto em que estava quando o último tesouro foi encontrado.

Caso o tesouro demore muito a ser encontrado, a linha 250 chamará a sub-rotina que se inicia na linha 500. Se, mesmo não tendo achado o tesouro, o homenzinho ainda dispuser de algum tempo, a linha 260 retornará ao programa, calculando sua nova posição.

A EXIBIÇÃO DO ESCORE

Esta é a sub-rotina final. Ela exibirá o escore e o número de "vidas", depois que uma "vida" for perdida:

```
500 CLS:SCREEN 0,0:LI=LI-1
510 PRINT @106,"NIVEL=";L$
520 IF LI>0 THEN PRINT @202,"VI
DAS=";LI
530 PRINT @298,"SCORE=";SC
540 IF LI>0 THEN FOR J=1 TO 600
0:NEXT:TIMER=0:SCREEN 1,1:RETUR
N
550 PRINT @358,"QUER OUTRA VEZ
(S/N)?"
```

Um pirata à procura de um fabuloso tesouro no labirinto do MSX.

Outro bucaneiro explora o colorido labirinto do Spectrum.

```
560 A$=INKEY$:IF A$<>"S" AND A$
<>"N" THEN 560
570 IF A$="S" THEN RUN
580 END
```

Agora você pode rodar o programa. Se uma "vida" for perdida, o programa religa a tela de texto — **SCREEN 0,0.** A linha 500 também diminui de 1 o número de "vidas" restantes do jogador.

As linhas 510 a 530 exibem o nível de dificuldade, o número de "vidas" restantes (se o jogo continuar) e a contagem de pontos. Se ainda sobrarem algumas "vidas", a linha 540 introduzirá uma pausa antes de zerar o cronômetro, religando a tela de alta resolução e retornando à sub-rotina.

As linhas 550 e 580 perguntam se o jogador quer tentar de novo, e ele ou pára o programa ou roda novamente. O **RUN** é utilizado na linha 570 para limpar P para um novo labirinto.

MSX

A versão do programa para os micros da linha MSX desenha os labirintos na tela com gráficos de alta resolução. Embora mais complexa do que o programa para se desenhar na tela de textos, ela tem a vantagem de poder traçar labirintos de diferentes complexidades, fornecendo assim uma variação no nível de dificuldade.

O caminho aleatório é formado por uma série de blocos quadrados. Se você quisesse desenhar um labirinto bem simples, bastaria escolher um bloco grande, ao passo que, se quisesse um labirinto mais complexo, teria que optar por um bloco de menor tamanho.

A primeira seção do programa introduz as variáveis e prepara o computador, de um modo geral, para desenhar labirintos aleatórios. Digite o programa, mas não o execute ainda, pois não aparecerá nada na tela, por enquanto.

```
10 SCREEN 0
20 LOCATE 2,11:PRINT "Nível de
dificuldade (0-5) ?"
30 L$=INKEY$:IF L$<"0" OR L$>"5
" THEN 30
40 BS=13-VAL(L$):NX=2*INT(.5+12
8/BS):NY=2*INT(.5+96/BS)
50 SX=250-BS*NX:SY=190-BS*NY
60 DIM P(NX,NY)
70 FOR I=1 TO 8
80 READ A,B,C:A$=A$+CHR$(A)
84 B$=B$+CHR$(B)
88 C$=C$+CHR$(C)
90 NEXT
100 CLS:LOCATE 2,11
105 PRINT "Gerando labirinto de
 nível ";L$
110 DATA 24,24,127,24,24,65,60,
255,93,82,255,85,82,24,81,24,24
,95,36,24,64,36,24,127
```

A linha 10 diz ao computador para utilizar a tela de textos (**SCREEN 0**). A linha 20 exibe, então, a mensagem: "Nível de dificuldade (0 - 5) ?"

L$** é o nível que o jogador escolheu. O valor numérico de **L$ — VAL (L$) na linha 40 — regula o tamanho do bloco, a extensão do caminho e a complexidade do labirinto. O INKEY$ na linha 30 significa que o jogador não precisa digitar mais que um único dígito em resposta à pergunta, e que o programa continua sem que ele precise pressionar **<ENTER>**.

Na linha 40, o BS é o tamanho do bloco gráfico, em pixels. O tamanho pode variar de oito a doze pixels. NX é o número de blocos na direção horizontal, e NY é o número de blocos na direção vertical.

Antes de desenhar o labirinto na tela, o computador calculará sua aparência final e colocará essa informação no conjunto P, que é dimensionado (**DIM**) na linha 60. A variável **A$** contém o formato do homenzinho, e a variável **B$**, uma cruz, para assinalar sua "morte". A variável **C$** contém a imagem do tesouro. Os códigos gráficos correspondentes são lidos em **DATA,** na linha 110, e concatenados nas variáveis alfanuméricas mencionadas. Posteriormente, esses blocos gráficos serão colocados dentro de sprites.

O computador limpa a tela de textos na linha 100, e esta informa que o labirinto está sendo produzido.

DESENHE O LABIRINTO

Agora, digite a parte do programa que produz o labirinto, e rode-o para ver o resultado:

```
1000 FOR J=0 TO NX
1005 P(J,NY)=6:P(J,0)=6
1008 NEXT
1010 FOR J=0 TO NY-2
1014 P(0,J)=6:P(NX,J)=6
1018 NEXT
1020 X=2:Y=2:LX=2:LY=2
1030 J=INT(RND(1)*4):G=J
1040 Y=LY+2*((J=0)-(J=2)):X=LX+
2*((J=3)-(J=1))
1050 IF P(X,Y)=0 THEN P(X,Y)=J+
1:P((X+LX)/2,(Y+LY)/2)=5:LX=X:L
Y=Y:GOTO 1030
1060 J=(J+1) AND 3:IF J<>G THEN
 1040
1070 J=P(LX,LY)-1:P(LX,LY)=5
1075 IF J<4 THEN LX=LX-2*((J=3)
-(J=1)):LY=LY-2*((J=0)-(J=2)):G
```

```
OTO 1030
1080 FOR J=0 TO 20
1082 P(2+2*INT(RND(1)*((NX-3)/2
)),1+INT(RND(1)*(NY-3)))=5
1085 P(1+INT(RND(1)*(NX-3)),2+2
*INT(RND(1)*((NY-3)/2)))=5
1088 NEXT
1090 SCREEN 2,0
1092 COLOR 1,10,4
1094 SPRITE$(1)= A$
1098 SPRITE$(0)= B$
1099 SPRITE$(2)=C$
1100 FOR J=2 TO NX-2
1105 FOR K=2 TO NY-2
1110 IF P(J,K)=5 THEN LINE (J*B
S+SX,K*BS+SY)-((J+1)*BS+SX-1,(K
+1)*BS+SY-1),4,BF
1130 NEXT K
1140 NEXT J
```

As linhas 1000 a 1080 "desenham" o labirinto na memória do computador e armazenam sua forma no conjunto P (cada elemento do conjunto corresponde a um bloco do labirinto). A sub-rotina armazena o número 5 em P onde quer que exista um caminho, e o número 0 faz o mesmo, caso exista uma parede.

Uma vez que o labirinto tenha sido armazenado em P, a linha 1090 liga a tela de alta resolução, e a linha 1092 define as cores de frente e de fundo do desenho. As linhas 1094 a 1099 definem os sprites correspondentes aos blocos gráficos montados na seção anterior do programa. Note que o sprite do bloco em branco tem prioridade sobre o bloco do homenzinho. As linhas 1100 a 1140 exibem o labirinto na tela mediante a verificação dos conteúdos de P. Quando um 5 for encontrado, um quadrado será impresso (comando **LINE** de bloco cheio) na linha 1110.

MONTE O JOGO

Agora você precisa de um jogo para divertir-se com o labirinto aleatório. Digite a próxima seção do programa; mas não o rode, pois ele chamará uma sub-rotina que ainda não existe.

```
1200 X=2:Y=2:LX=2:LY=2
1210 TI=800:LI=3
1215 R=RND(-TIME)
1220 TIME=0
1230 X1=1+INT(RND(1)*(NX-3))
1240 Y1=1+INT(RND(1)*(NY-3))
1250 IF P(X1,Y1)=5 THEN P(X1,Y1
)=7:PUT SPRITE 2,(SX+X1*BS,SY+Y
1*BS),10 ELSE 1230
1255 X1=X*BS+SX:Y1=Y*BS+SY
1260 PUT SPRITE 1,(X1,Y1),1
1270 T$=INKEY$:IF T$="" THEN 12
70
1280 IF ASC(T$)=28 THEN X=X+1
1290 IF ASC(T$)=29 THEN X=X-1
1300 IF ASC(T$)=30 THEN Y=Y-1
1310 IF ASC(T$)=31 THEN Y=Y+1
1320 IF P(X,Y)=7 THEN F=1:P(X,Y
)=5:GOTO 1330
1325 IF P(X,Y)<>5 THEN X=LX:Y=L
Y:GOTO 1270
1330 IF X<>LX OR Y<>LY THEN LX=
X:LY=Y
1340 IF F=1 THEN F=0:SC=SC+(TI-
TIME):TI=TI-10:GOTO 1220
1350 IF TIME>TI THEN GOSUB 1500
:IF LI<1 THEN 1000
1360 GOTO 1255
```

A linha 1200 contém as variáveis a partir das quais a posição do homenzinho é calculada. X e Y são as coordenadas da posição atual do homenzinho, e LX e LY, da última posição. **TI** é o limite de tempo para descobrir o tesouro, e LI, o número de "vidas" de que o jogador dispõe. O tempo limite é de cerca de dezesseis segundos; se este se esgotar sem que o tesouro tenha sido encontrado, o homenzinho perderá uma "vida". No início, o jogador tem três "vidas".

A linha 1215 introduz o gerador de números aleatórios, usando como "semente" o valor armazenado no cronômetro interno, naquele instante. O cronômetro interno do computador é zerado na linha 1220, antes que as linhas 1230 e 1240 selecionem uma posição aleatória para o tesouro. O elemento correspondente em P é examinado na linha 1250, para verificar se nesse lugar há um caminho ou uma parede. Se houver um caminho, o valor do conjunto será modificado de 5 para 7. A última parte da linha desenha o tesouro no labirinto.

A posição do pirata é calculada na linha 1255, levando-se em conta a largura do caminho, que é o tamanho do bloco BS. A linha 1260 coloca o homenzinho na tela, naquela posição.

As linhas 1270 a 1310 examinam o teclado através do comando **INKEY$**, de modo a permitir a movimentação contínua do homenzinho, no labirinto, comandado pelas teclas de controle do cursor. A linha 1325 faz com que ele se mantenha dentro do caminho, de modo a impedi-lo de atravessar alguma parede. A linha 1320 checa se o tesouro foi encontrado, examinando o elemento correspondente em P para o número 7. Quando o tesouro é encontrado, o "indicador de descoberta" (F) é igualado a 1.

A linha 1330 promove a movimentação do homenzinho, fazendo com que a localização atual torne-se a última posição. Como o homenzinho é definido por intermédio de um sprite, não é necessário apagá-lo do lugar onde estava, como em outros computadores.

A linha 1340 calculará o escore, se o tesouro for encontrado. O limite de tempo é diminuído de 10 segundos. O F retorna ao zero e o programa volta à linha 1220 para zerar o cronômetro. O programa continua, redesenhando o tesouro em outro lugar, mas deixando o pirata no ponto em que ele estava quando o último tesouro foi encontrado.

Se o jogador demorar muito para achar o tesouro, a linha 1350 chamará a sub-rotina que se inicia na linha 1500 (ela será apresentada a seguir). Essa sub-rotina diminui uma "vida" do jogador; caso tenha se esgotado o número de "existências", o programa retornará ao começo (linha 1000). Se o homenzinho não tiver achado o tesouro e ainda dispuser de algum tempo, a linha 1360 fará o programa voltar à linha 1255, permitindo que sua nova posição seja calculada.

EXIBA O ESCORE

A sub-rotina final exibirá o escore e o número de "vidas" restantes depois que uma "vida" for perdida:

```
1500 LI=LI-1
1505 PUT SPRITE 0,(X1,Y1),1
1507 OPEN "GRP:" FOR OUTPUT AS
#1
1510 PSET(0,0),10:PRINT #1,TAB(
3);"VIDAS:";LI;TAB(5);"SCORE =
";SC
1520 FOR J=1 TO 6000:NEXT
1525 LINE(0,0)-(255,7),10,BF
1530 IF LI>0 THEN PUT SPRITE 0,
(-20,-20):CLOSE #1:TIME=0:RETUR
N
1540 PSET(0,0),4:PRINT #1," Que
r jogar novamente ? (S/N)"
1550 A$=INKEY$:IF A$<>"S" AND A
$<>"N" THEN 1550
1560 IF A$="S" THEN RUN
1570 END
```

Rode o programa: se uma "vida" for perdida, ele diminuirá de 1 o número de "existências" restantes (linha 1500). A linha 1505 assinala a "morte" do personagem.

As linhas 1510 a 1530 exibem o nível de dificuldade, o número de "vidas" restantes e a contagem de pontos. Caso ainda sobrem "vidas", a linha 1530 colocará o cronômetro em 0.

As linhas 1540 a 1560 perguntam se o jogador quer tentar de novo; dependendo da resposta, elas param o programa ou o rodam novamente. O **RUN** é utilizado na linha 1560, limpando P para um novo labirinto.

UMA TÉCNICA DE ANIMAÇÃO DE BLOCOS GRÁFICOS NOS MICROCOMPUTADORES DA LINHA TRS-80.

Os micros da linha TRS-80, por serem de tecnologia mais antiga do que os da nova geração de computadores pessoais (Sinclair Spectrum, MSX, TRS-Color), contam com menores recursos para a definição de blocos gráficos (UDG, ou user = defined graphics*) além de terem uma resolução gráfica na tela (128 x 48 pixels) insuficiente para a programação de efeitos visuais mais sofisticados.*

Existem, porém, formas de contornar esse problema. Assim, para definir e animar blocos gráficos no TRS-80, utilizamos três conceitos básicos de programação: como incorporar caracteres gráficos a variáveis alfanuméricas (cordões); como usar os caracteres de controle do cursor para definir figuras complexas, e como movimentar blocos gráficos por meio do comando **PRINT@**.

Como a utilização desse comando já foi explicada, abordaremos aqui apenas a técnica de definição de figuras complexas.

Imagine que queremos definir um disco voador. Para montar a figura em uma única variável alfanumérica (por exemplo, **D$***) concatenamos os caracteres gráficos que determinam a primeira linha (veja no manual do seu computador os códigos gráficos correspondentes):*

```
A$ = CHR$(128)+CHR$(186)+
     CHR$(176)+CHR$(186)+
     CHR$(144)
```

Acrescentamos a seguir a linha do meio da figura. Mas, se nos limitarmos a concatenar os próximos caracteres gráficos ao **D$** *já definido, na hora de colocar o disco voador na tela por meio de um* **PRINT@**, *o desenho sairá errado (em vez de se posicionarem uma embaixo da outra, as duas linhas ficarão fundidas em uma única linha). Para evitar isso, temos que fazer com que o cursor se posicione no ponto certo, abaixo da primeira linha do gráfico. Para isso, usamos os códigos de controle:* **CHR$(26)**, *que tem o efeito de descer o cursor do* **PRINT** *de uma posição na tela, e* **CHR$(24)**, *que recua o cursor, sem limpeza.*

Dessa forma, para posicionar o cursor corretamente na linha seguinte do gráfico, temos que descer uma posição com o cursor e recuar cinco posições. Para economizar tempo de digitação, podemos definir um cordão **B$**, *que incorpora todos esses* **CHR$**:

```
B$ = CHR$(26)+STRING$(5,24)
```

E agora **A$** *fica definido assim:*

```
A$ = CHR$(128)+CHR$(186)+
     CHR$(176)+CHR$(186)+
     CHR$(144)+B$+CHR$(170)+
     STRING$(4,171)
```

Finalmente, podemos completar a figura, adicionando a terceira linha de blocos gráficos, e repetindo o truque de descer uma posição e recuar o cursor cinco posições:

```
A$ = CHR$(128)+CHR$(186)+
     CHR$(176)+CHR$(186)+
     CHR$(144)+B$+CHR$(170)+
     STRING$(4,171)+B$+
     CHR$(128)+STRING$(3,131)+
     CHR$(129)
```

O bloco gráfico está definido. Para desenhar, de uma vez só, esse bloco em um ponto qualquer da tela (por exemplo, na posição N), basta dar o comando:

```
PRINT@N,A$;
```

Para animar o bloco gráfico contido em **A$**, *variamos o valor de N, colocando-o dentro de um laço de programa. Se o valor de N for aumentado ou diminuído de 64, em cada repetição do laço, a movimentação se dará no sentido vertical. Por outro lado, se esse valor for aumentado ou diminuído de 1, o movimento se fará no sentido horizontal.*

Para apagar o bloco gráfico do lugar onde estava, é necessário colocar por cima dele um terceiro bloco gráfico previamente definido, contendo apenas três linhas de cinco caracteres em branco, concatenadas por meio do cordão de controle, **B$**, *definido acima.*

E AGORA...O QUE FAZER?

- O COMANDO **INPUT** E AS MENSAGENS DE PRONTIDÃO
- MÉTODOS MAIS RÁPIDOS COM **INKEY$** OU **GET**
- PROGRAMAS DE DESENHOS NA TELA
- ENTRADAS DE SENHAS SECRETAS

Grande parte dos programas para micros exige que o usuário digite dados e outras informações necessárias ao seu funcionamento. A maneira como isso é feito tem uma importância fundamental. Veja por quê.

Com muito poucas exceções, os computadores não funcionam inteiramente sozinhos, dando saída a informações. Quase todos os programas, desde jogos até aplicações comerciais e científicas, exigem do usuário algum tipo de entrada. Entretanto, há informações que podem ser fornecidas diretamente ao computador, e existem inúmeras maneiras pelas quais isso pode ser feito, para os vários tipos de programa.

A informação pode ser tão simples quanto a pressão sobre uma tecla de controle do cursor para direcionar uma base de mísseis. Mais comumente, porém, consiste na digitação de números ou textos como acontece, por exemplo, em programas para bancos de dados. No caso de um jogo, a maior exigência é que o computador responda de modo rápido.

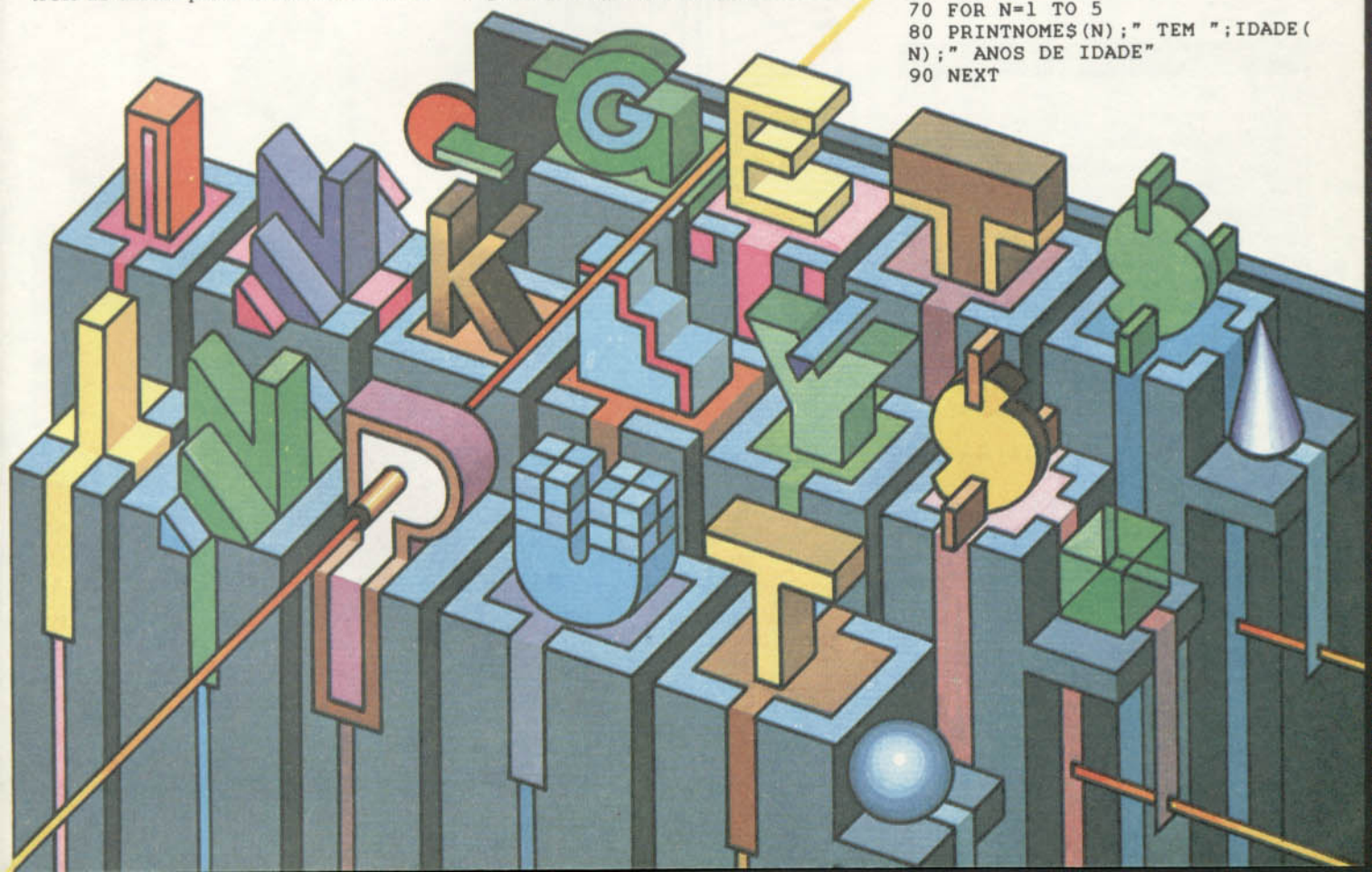
INPUTS SIMPLES

Um dos primeiros programas executados pelos iniciantes e que ilustra o uso do comando **INPUT** é o seguinte:

```
10 PRINT"QUAL E O SEU NOME?"
20 INPUT N$
30 PRINT"OLA, ";N$
```

Ao deparar-se com um **INPUT** dentro de um programa, o computador pára o processamento e espera até que o usuário digite algo. Após terem sido pressionados **<ENTER>** ou **<RETURN>**, a mensagem digitada (qualquer que seja ela) é colocada em uma variável — **N$**, no caso do programa acima.

Neste exemplo, a variável é um cordão alfanumérico, mas variáveis numéricas também podem ser utilizadas. No programa seguinte, mostraremos como se faz uma lista de cinco nomes e idades, empregando comandos INPUT simples. Esse programa utiliza dois conjuntos para armazenar informações (mais adiante, abordaremos numa lição específica a questão dos conjuntos e de como funcionam).

```
5 DIM NOMES$(5),IDADE(5)
10 FOR N=1 TO 5
20 PRINT"NOME:"
30 INPUT NOMES$(N)
40 PRINT"IDADE:"
50 INPUT IDADE(N)
60 NEXT N
70 FOR N=1 TO 5
80 PRINTNOMES$(N);" TEM ";IDADE(
N);" ANOS DE IDADE"
90 NEXT
```

```
5  DIM n$(5,10): DIM a(5)
10  FOR n=1 TO 5
20 PRINT "Nome: "
30 INPUT n$(n)
40 PRINT "Idade: "
50 INPUT a(n)
55 CLS : NEXT n
60 FOR n=1 TO 5
70 PRINT n$(n);" tem ";a(n);"
 anos de idade"
80 NEXT n
```

Nos computadores da linha ZX-81, digite tudo com letras maiúsculas. Omita as linhas 5 e 55, e acrescente:

```
5 DIM N$(5,10)
7 DIM A(5)
55 CLS
57 NEXT N
```

A informação fornecida ao usuário deve estar correta. Da mesma forma, o usuário precisa entrar exatamente o tipo de informação solicitada. Se você digitar um nome quando for perguntada uma idade, o programa será interrompido. No entanto, rodar o programa mais uma vez leva quase sempre à perda de tudo o que já foi entrado, obrigando o operador a um duplo trabalho. Isso não constitui problema se forem apenas cinco entradas; mas quando se trata de um programa extenso, esse esforço pode ser desgastante.

UTILIZE MENSAGENS DE PRONTIDÃO

Embora pareça simples, o programa acima tem uma característica importante, que é a *mensagem de prontidão* contida nas duas declarações **PRINT**. Essa mensagem faz com que o programa diga ao usuário que tipo de informação ele deve digitar pelo teclado.

Se as linhas 20 e 40 do programa fossem apagadas, ainda assim ele funcionaria, pois, na maioria dos computadores, toda ocorrência de um **INPUT** provoca o aparecimento de algum sinal de prontidão na tela (por exemplo, um ponto de interrogação).

Se você já está familiarizado com computadores, o sinal de prontidão do **INPUT** é suficiente, quase sempre, para lembrá-lo de que alguma coisa precisa ser entrada. Ele é insuficiente, porém, para lhe dizer qual o tipo de informação a ser digitada. Imagine, por exemplo, que o programa lhe é estranho, que talvez você não tenha experiência em lidar com o computador ou que não tenha conhecimento de como ele funciona. Neste caso, um ponto de interrogação causa apenas estupefação...

A mensagem de prontidão é necessária até mesmo nos programas mais simples, pois o usuário tende a esquecer facilmente a forma exata pela qual a informação deve ser entrada. Por exemplo, a entrada de datas. A maior parte dos programas de contabilidade, além de muitos outros, requer a entrada de datas pelo usuário. Em alguns casos, essa informação é utilizada pelo programa, seja em rotinas de pesquisa, quando se deseja encontrar um dado específico, seja na ordenação cronológica de uma série de eventos, por exemplo. Neste caso, é preciso estabelecer que uma data seja sempre entrada segundo uma forma padronizada.

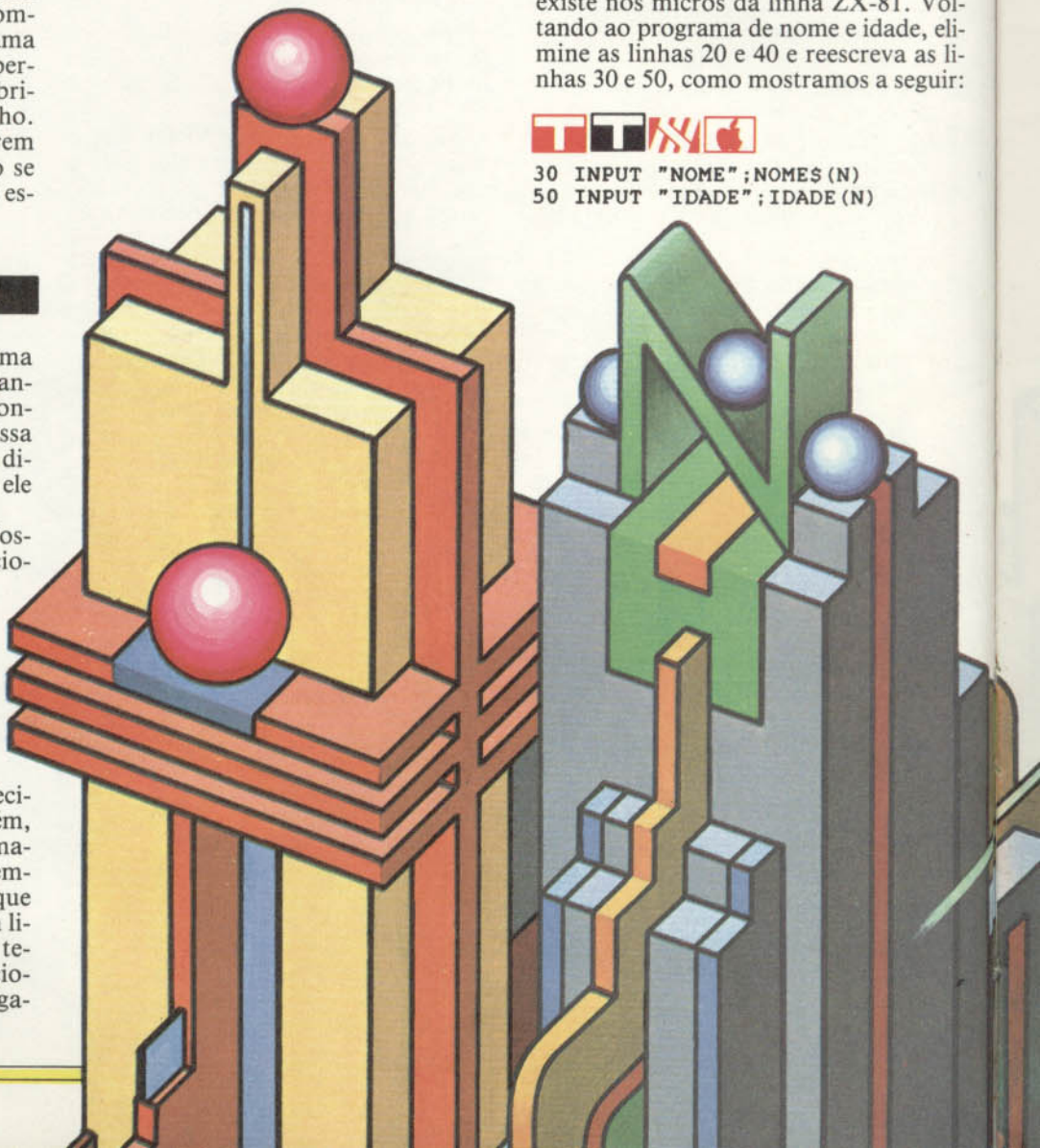
Um método comum é utilizar seis dígitos para uma data, ou seja, dois dígitos para o dia, dois para o mês, e dois para o ano. Muitas vezes empregam-se barras ou pontos para separar os três períodos. Uma rotina típica de entrada de dados através de um **INPUT** seria:

```
100 PRINT "DIGITE DATA (EM FORM
ATO DD/MM/AA)"
110 INPUT D$
```

A mensagem serve não somente para lembrar o usuário de que uma data é necessária, como também para definir a forma como ela deve ser entrada (o que se denomina *formato de entrada* do dado). Se você quiser entrar, por exemplo, 3 de setembro de 1945, deverá digitar: 03/09/45.

O formato de entrada poderá ser qualquer um que o programa desejar, desde que o usuário seja informado, por meio de uma mensagem adequada.

A utilização do comando **INPUT**, como foi mostrada até agora, é bastante simples. Mas ela pode ser ainda mais simplificada no caso de alguns computadores, que permitem a inclusão da mensagem de prontidão no próprio comando **INPUT**. Esse recurso só não existe nos micros da linha ZX-81. Voltando ao programa de nome e idade, elimine as linhas 20 e 40 e reescreva as linhas 30 e 50, como mostramos a seguir:

```
30 INPUT "NOME";NOME$(N)
50 INPUT "IDADE";IDADE(N)
```

Na maioria dos micros (com exceção do TRS-Color), se não for colocado um espaço em branco antes do segundo sinal de aspas, na mensagem de prontidão, o ponto de interrogação aparecerá colado a ela. Por isso, se você quiser dar maior clareza à mensagem, coloque o espaço em branco adicional.

Com exceção do Sinclair Spectrum, é possível compactar ainda mais a rotina de entrada de dados. Assim, as linhas 30 e 50 do programa anterior poderiam ser substituídas por apenas uma:

```
30 INPUT"DIGITE NOME E IDADE ";
NOME$(N),IDADE(N)
```

(Não se esqueça de apagar a linha 50.)

No Spectrum, é necessário colocar somente uma variável por comando **INPUT**, de modo que o usuário deve pressionar duas vezes o <**ENTER**>. No caso do TRS-Color, do TRS-80, do MSX e do Apple II, além de se poder utilizar o método do Spectrum, a linha única oferece a possibilidade de se digitar nome e idade separados apenas por uma vírgula, e de se pressionar o <**ENTER**> somente no final. Pode-se ainda colocar qualquer número de variáveis — separadas por vírgulas em uma única linha de **INPUT**.

Ao se utilizar apenas uma linha de **INPUT** para várias entradas, não se deve esquecer de incluir instruções sobre todas as informações pedidas, bem como os seus formatos. Além disso, é necessário lembrar o usuário de que os dados devem ser separados uns dos outros por vírgulas, e que só no final se deve pressionar a tecla <**ENTER**>. Se essa norma não é seguida a maioria dos micros interrompe o programa, colocando um duplo sinal de interrogação na linha seguinte.

No Sinclair é possível dividir a mensagem de prontidão em mais de uma, na mesma linha. Se cada mensagem de prontidão for mantida entre aspas, o computador se limitará a imprimi-las, sem tratá-las como variáveis.

Entretanto, note que não é possível fazer isso nos computadores das linhas TRS-Color, Apple, TRS-80, MSX e ZX-81. Para melhorar a disposição das informações na tela, as mensagens de prontidão dos **INPUT** podem ser posicionadas por intermédio de comandos **TAB** normais (conforme foi explicado anteriormente, na lição sobre o uso dessa função).

COMO ENTRAR UMA LINHA COMPLETA

O principal problema na utilização do comando **INPUT** surge quando se tenta entrar cordões alfanuméricos que contenham vírgulas e dois pontos, ou espaços em branco no início. Um endereço, por exemplo, normalmente contém vírgulas, de modo que fica impossível usar o **INPUT** para colocá-lo em uma variável alfanumérica simples. Do mesmo modo, a inclusão de espaços no início de uma entrada pode ser útil para a beleza da apresentação na tela.

A dificuldade está em que muitos computadores ignoram espaços à esquerda em um comando **INPUT**, embora não criem problemas com os intervalos entre as palavras. E, quase sempre, eles também ignoram qualquer coisa que ocorra após vírgula, dois pontos ou ponto e vírgula, truncando a informação restante, mesmo que o usuário a tenha digitado. Felizmente, alguns micros possuem um comando que contorna essa dificuldade. Os computadores da linha Sinclair, por sua vez, têm um modo especial de evitá-la.

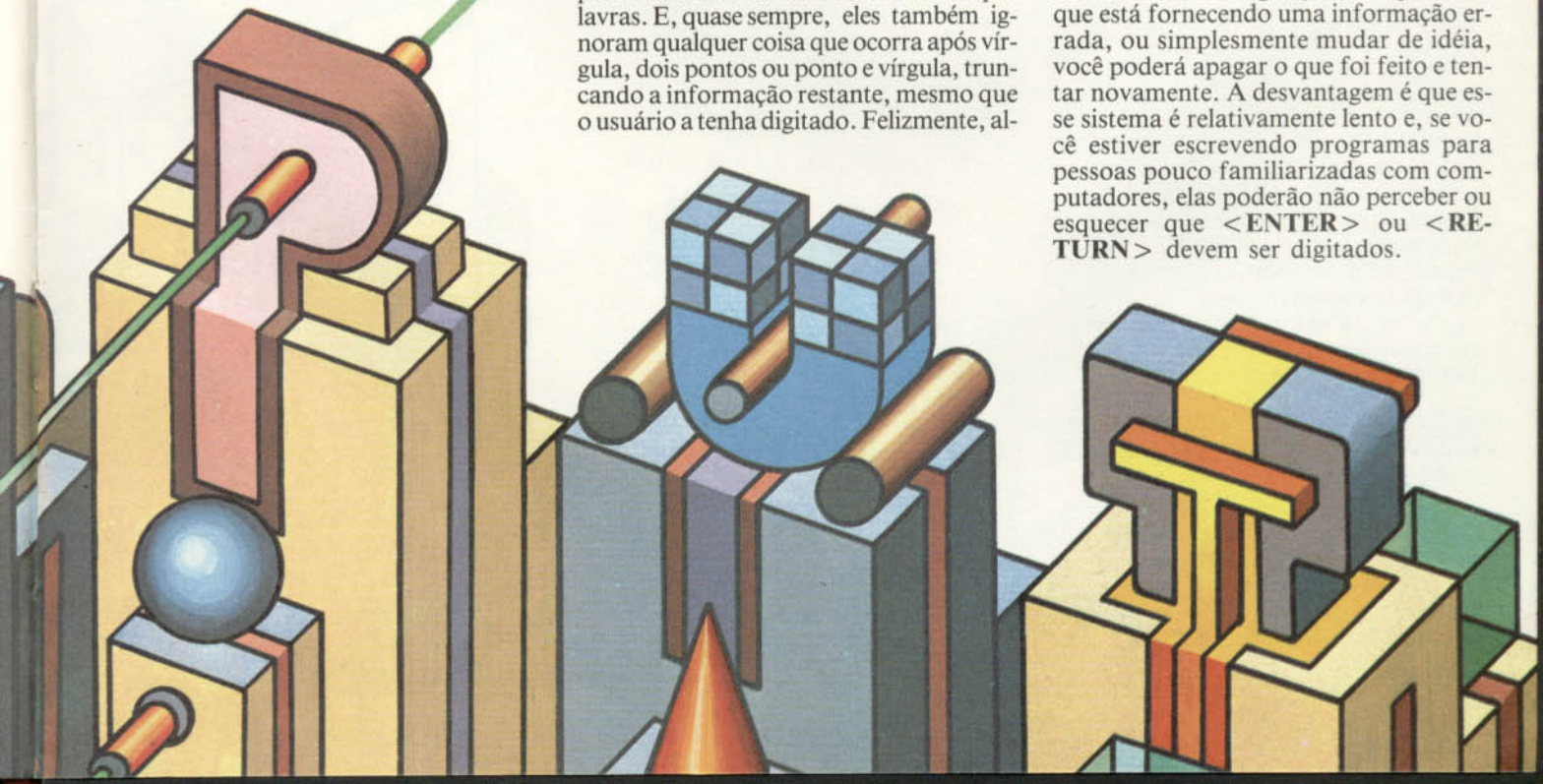
A solução mais comum é incluir a entrada entre aspas. O ZX-81 e o Spectrum, por exemplo, colocam aspas automaticamente na tela quando uma variável alfanumérica aparece em um comando **INPUT**. Em relação aos outros computadores, o operador deve digitar as aspas como parte da entrada (essa exigência, portanto, deve ser apontada ao usuário, através da mensagem de prontidão).

Uma outra solução, disponível nos computadores TRS-80, TRS-Color e MSX, consiste em utilizar o comando **LINE INPUT**. Este é empregado exatamente do mesmo modo que o **INPUT**.

```
10 LINE INPUT "POR FAVOR, DIGIT
E O SEU ENDERECO ";A$
```

A vantagem é que se pode colocar, dentro da variável, qualquer coisa até o <**RETURN**>, incluindo vírgulas e espaços. Isto é mais garantido, pois você não precisará lembrar o usuário a digitar entre aspas.

ACELERE AS ENTRADAS

Uma das qualidades do comando **LINE INPUT** consiste em permitir que você altere o que estiver entrando, até o momento de pressionar <**RETURN**> ou <**ENTER**>. Assim, se você cometer um erro de digitação, ou perceber que está fornecendo uma informação errada, ou simplesmente mudar de idéia, você poderá apagar o que foi feito e tentar novamente. A desvantagem é que esse sistema é relativamente lento e, se você estiver escrevendo programas para pessoas pouco familiarizadas com computadores, elas poderão não perceber ou esquecer que <**ENTER**> ou <**RETURN**> devem ser digitados.

Em programas que utilizam uma porção de menus ou respostas do tipo "sim ou não", a necessidade de pressionar uma tecla extra diminui a velocidade de execução, além de praticamente duplicar o número de teclas a serem pressionadas. Para evitar que isto aconteça, existe uma maneira de levar o computador a detectar automaticamente a tecla que está sendo pressionada, sem que seja necessário acionar depois as teclas <ENTER> ou <RETURN>

O comando utilizado é o **INKEY$** nos computadores Sinclair, TRS-80, TRS-Color e MSX, e o **GET** nos da linha Apple II.

O efeito da função **INKEY$**, que já estudamos anteriormente, é provocar uma varredura no teclado para averiguar se alguma tecla está sendo pressionada nesse instante. Se este for o caso, a função trará o caractere correspondente de volta ao programa principal. No caso de nenhuma tecla estar sendo pressionada no momento em que o **INKEY$** é encontrado, um cordão vazio é trazido de volta. Assim o **INKEY$** é usado dentro de um comando **IF**, que testa continuamente se o valor retornado é diferente do vazio:

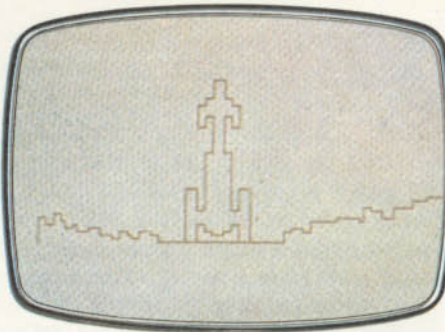
```
100 LET A$=INKEY$:IF A$="" THEN
 GOTO 100
```

Já nos micros da linha Apple II, o **GET** faz o computador esperar até que a tecla seja pressionada, não sendo necessário colocá-lo dentro de um laço de espera, como nos outros casos:

```
100 GET A$
```

Qualquer variável alfanumérica pode ser usada — **A$** é apenas um exemplo. A máquina pára na linha 100. Se você teclar o R, o **A$** conterá a letra R. Você pode entrar um número, ou um espaço, ou qualquer outro caractere, até mesmo teclas de controle como <**ENTER**>. Embora seja virtualmente qualquer coisa, a entrada pode ter apenas um caractere. Assim que a tecla for pressionada, o programa prosseguirá.

DESENHE NA TELA

Agora examine o seguinte programa, que utiliza quatro teclas para desenhar em alta resolução na tela (esse programa não funciona no ZX-81 e no TRS-80, que não dispõem de alta resolução gráfica). Para desenhar no Apple II e no TRS-Color são utilizadas as teclas, Z, X, P e L; no MSX, as teclas a empregar são as de controle do cursor. Você também pode usar a tecla 2 para pedir um desenho na cor de fundo da tela (criando, portanto, uma linha invisível), ou a tecla 1, para retornar à linha do desenho visível.

Uma linha pode ser movimentada em diversas direções. Para isso, basta acionar algumas teclas indicadas pelo programa.

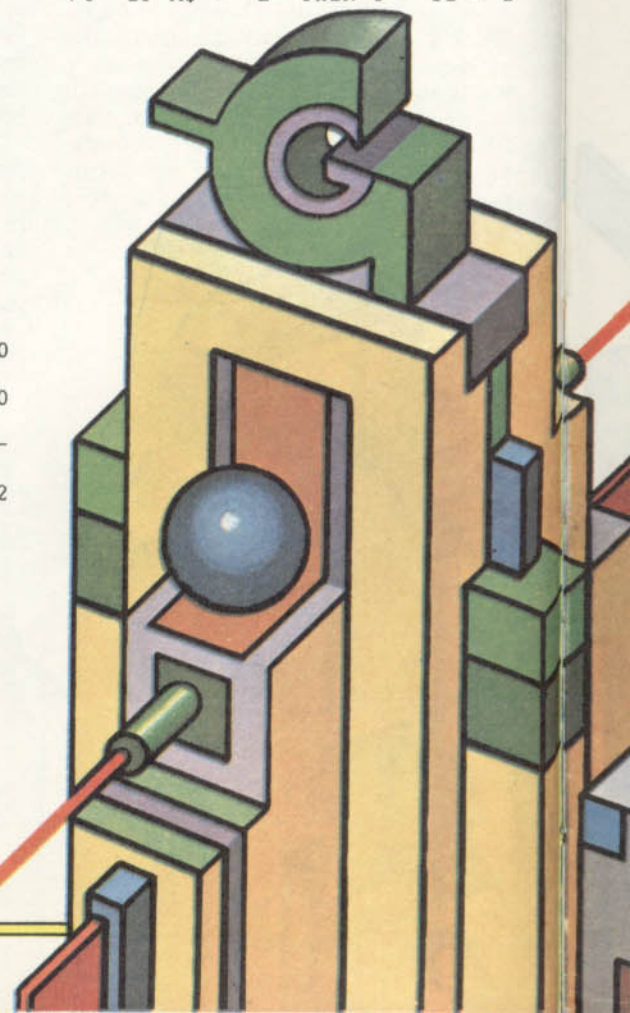
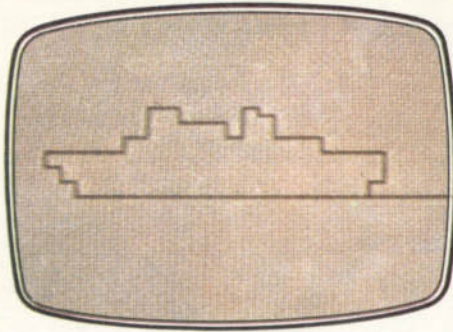
Com um programa simples, que controla apenas algumas teclas, é possível desenhar imagens como as da ilustração.

```
10 PMODE 4,1:PCLS:SCREEN 1,1
20 LET X=100:LET Y=100
40 LET X1=X:LET Y1=Y
50 LET A$=INKEY$
60 IF A$="P" THEN LET Y=Y1-4
70 IF A$="L" THEN LET Y=Y1+4
80 IF A$="Z" THEN LET X=X1-4
90 IF A$="X" THEN LET X=X1+4
100 IF A$="1" THEN COLOR 5
110 IF A$="2" THEN COLOR 0
120 IF A$=" " THEN STOP
130 LINE(X,Y)-(X1,Y1),PSET
140 GOTO 40
```

```
 10 INK 2
 20 PLOT 127,87
 30 IF INKEY$="p" THEN  DRAW 0
,2
 40 IF INKEY$="l" THEN  DRAW 0
,-2
 50 IF INKEY$="z" THEN  DRAW -
2,0
 60 IF INKEY$="x" THEN  DRAW 2
,0
 70 IF INKEY$="1" THEN  INK 2
 80 IF INKEY$="2" THEN  INK 7
 90 IF INKEY$=" " THEN  STOP
100 PAUSE 10
110 GOTO 30
```

```
10 SCREEN2
20 LETX=100:LETY=100
30 LETX1=X:LETY1=Y
40 A$=INKEY$
50 IFA$=CHR$(30)THENY=Y1-4
60 IFA$=CHR$(31)THENY=Y1+4
70 IFA$=CHR$(29)THENX=X1-4
80 IFA$=CHR$(28)THENX=X1+4
90 IFA$="1"THENCOLOR 4,4
100 IFA$="2"THENCOLOR 15,4
110 IFA$=CHR$(32)THENSTOP
120 LINE(X,Y)-(X1,Y1)
130 GOTO30
```

```
10  ONERR  GOTO 150
20  HGR : HCOLOR= 7
30 X = 100:Y = 100
40 X1 = X:Y1 = Y
50  GET A$
60  IF A$ = "P" THEN Y = Y1 - 2

70  IF A$ = "L" THEN Y = Y1 + 2
```

Um barco, um míssil, uma garrafa: se você quiser executar desenhos como esses, siga as instruções desenvolvidas no texto.

**Trace as diagonais como se fossem uma escada de minúsculos degraus.
Não pressione duas teclas ao mesmo tempo.**

```
80   IF A$ = "Z" THEN X = X1 - 2
90   IF A$ = "X" THEN X = X1 + 2
100  IF A$ = "1" THEN  HCOLOR=
7
110  IF A$ = "2" THEN  HCOLOR=
0
120  IF A$ = " " THEN  END
130  HPLOT X,Y TO X1,Y1
140  GOTO 40
150 X = X1:Y = Y1: PRINT  CHR$
(7);: RESUME
```

Esse tipo de rotina é muito útil para gráficos e jogos. A linha é traçada enquanto as teclas são pressionadas. Mas observe que o computador aceita uma tecla de cada vez; já as linhas diagonais são de traçado mais difícil, pois consistem em uma série de pequenos degraus.

Os comandos **INKEY$** ou **GET** são também muito úteis em qualquer programa que empregue menus. O programa seguinte imprime um menu como parte de um programa de arquivo de dados.

```
5 CLS
10 DATA CRIAR UM ARQUIVO,ENTRAR
 DADOS, VER DADOS,EDITAR,PROCUR
AR,IMPRIMIR,CARREGAR,GRAVAR,FIM
15 RESTORE
20 FORN=1TO9
30 READ OPCOES$
40 PRINTTAB(5);N;TAB(10);OPCOES
$
50 NEXTN
60 PRINT:PRINTTAB(5)"SUA OPÇÃO
=>"
70 A$=INKEY$:IFA$=""THEN70
80 IFA$="1"THENGOSUB1000
90 IFA$="2"THENGOSUB2000
100 IFA$="3"THENGOSUB3000
110 IFA$="4"THENGOSUB4000
120 IFA$="5"THENGOSUB5000
130 IFA$="6"THENGOSUB6000
140 IFA$="7"THENGOSUB7000
150 IFA$="8"THENGOSUB8000
160 IFA$="9"THENGOSUB9000
170 GOTO5
```

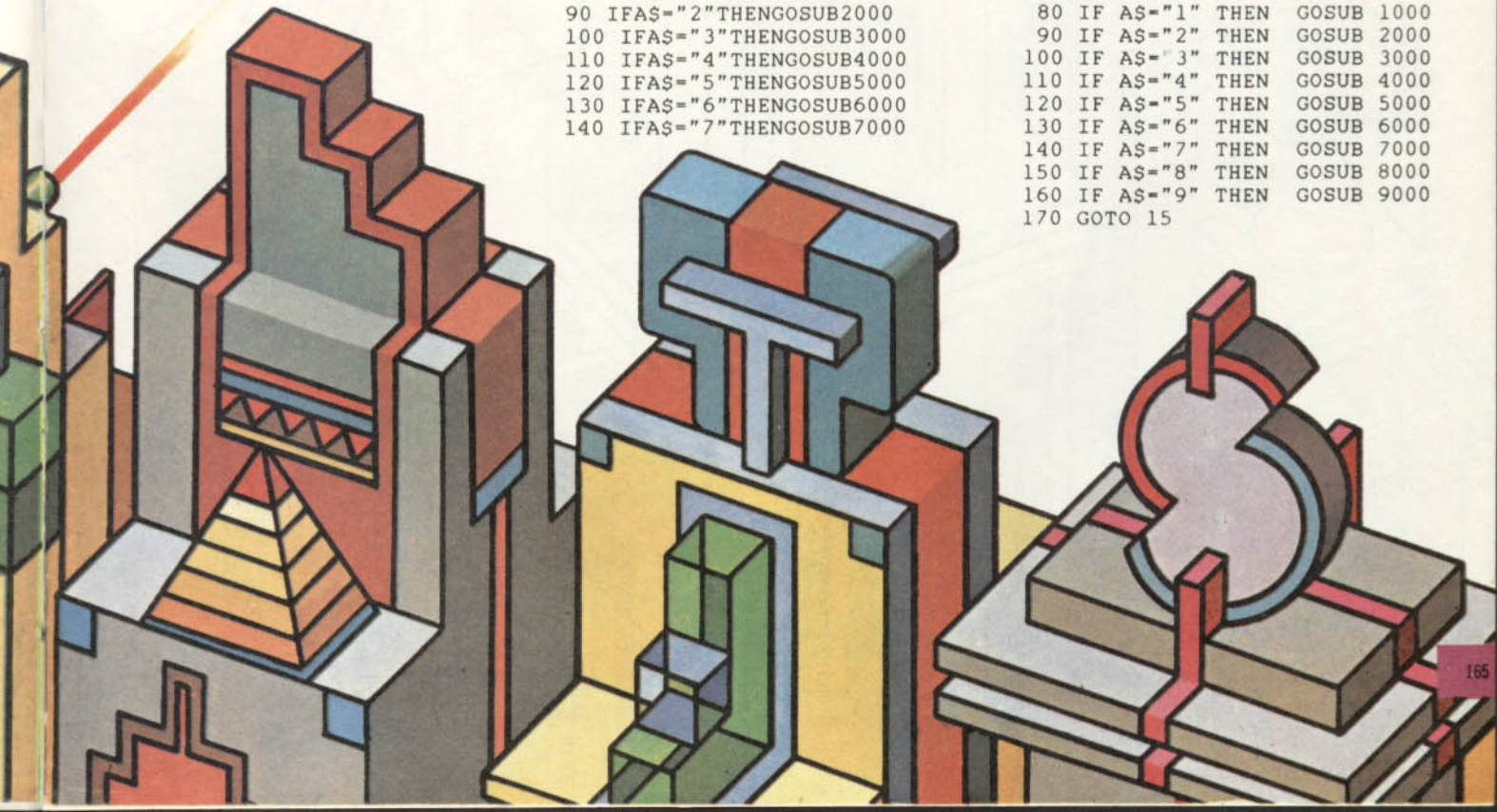
Altere a linha 70 do programa anterior para:

```
70 LET A$=INKEY$:IF A$="" THEN
GOTO 70
```

Modifique as linhas do programa anterior para:

```
5 HOME
70 GET A$:IF A$="" THEN 70
```

Apague as linhas 10 e 70 do programa anterior, substituindo-as por:

```
 10 DATA "Criar novo arquivo",
 "Entrar com registros","Ver r
 egistros","Editar registros",
 "Procurar registros","Imprimi
 r arquivo","Carregar arquivo"
 ,"Gravar arquivo","Saida"
 15 RESTORE
 20 FOR n=1 TO 9
 30 READ h$
 40 PRINT TAB 5;n;TAB 10;h$
 50 NEXT n
 60 PRINT : PRINT TAB 5;"Sua e
scolha->"
 70 LET A$=INKEY$: IF A$=""
 THEN  GOTO 70
 80 IF A$="1" THEN  GOSUB 1000
 90 IF A$="2" THEN  GOSUB 2000
100 IF A$="3" THEN  GOSUB 3000
110 IF A$="4" THEN  GOSUB 4000
120 IF A$="5" THEN  GOSUB 5000
130 IF A$="6" THEN  GOSUB 6000
140 IF A$="7" THEN  GOSUB 7000
150 IF A$="8" THEN  GOSUB 8000
160 IF A$="9" THEN  GOSUB 9000
170 GOTO 15
```

Com esse programa, o computador vai direto à sub-rotina pertinente assim que uma tecla é acionada — a menos que você pressione uma outra tecla entre os números de 1 a 9. Neste caso, a linha 170 imprimirá novamente o menu e este se repetirá na tela.

UMA ROTINA PARA SENHAS

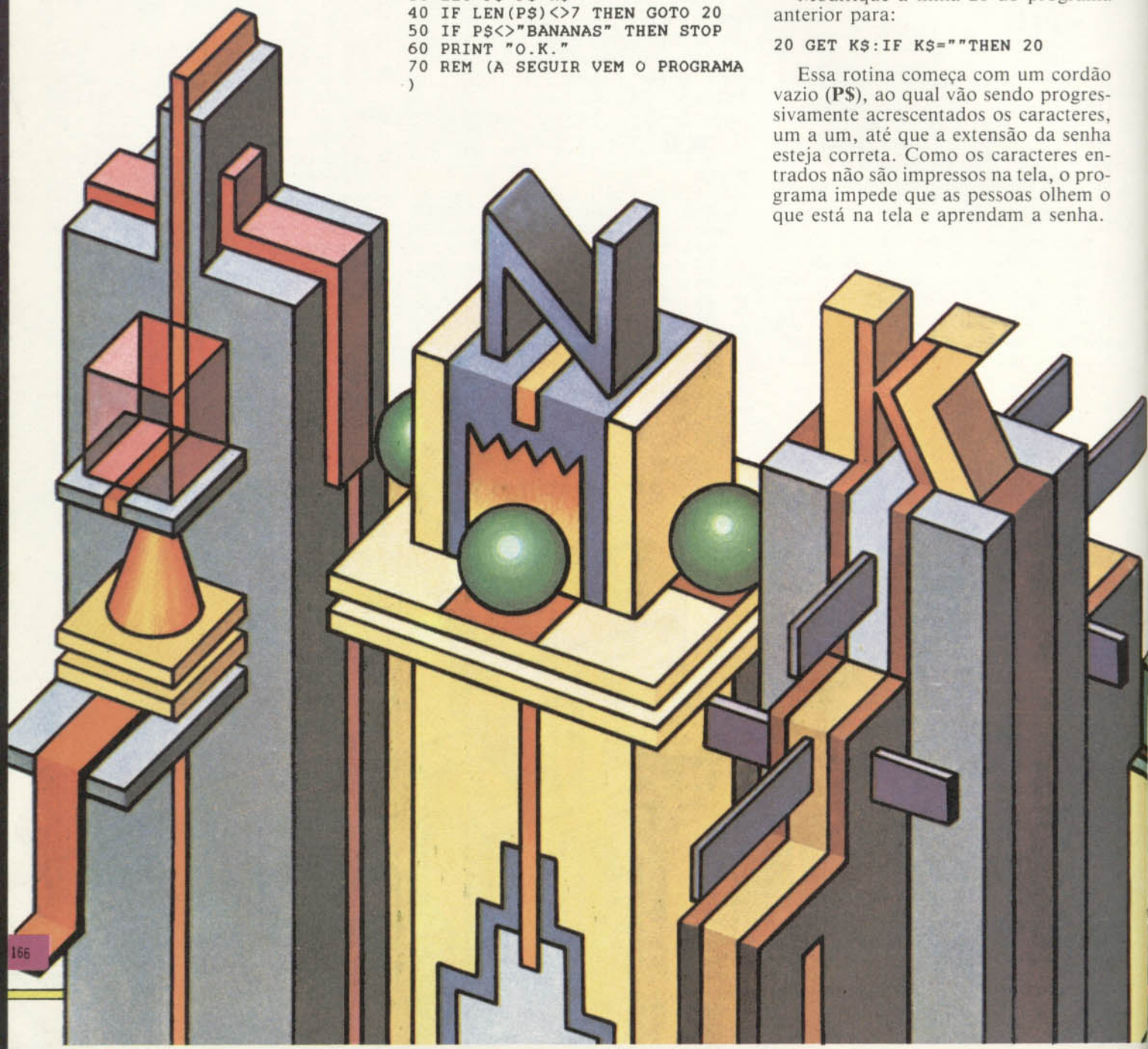
O programa anterior funcionará perfeitamente enquanto existirem menos de nove opções. Mas se você tentar teclar 10, o computador entenderá que você está entrando o 1, pois o programa prosseguirá automaticamente assim que tiver sido teclado o primeiro dígito. Mas existe uma maneira de entrar palavras inteiras. E como não é mostrado nada na tela, ela é ideal para se entrar uma senha ou um "código secreto", que pode ser usado para limitar o acesso dos usuários a um determinado programa. Cada dígito é entrado utilizando-se o **INKEY$** ou o **GET**, sendo o caractere resultante acrescentado ao último dígito entrado. Eis o programa:

```
10 PRINT "DIGITE A SENHA"
20 LET K$=INKEY$:IF K$="" THEN
GOTO 20
30 LET P$=P$+K$
40 IF LEN(P$)<>7 THEN GOTO 20
50 IF P$<>"BANANAS" THEN STOP
60 PRINT "O.K."
70 REM (A SEGUIR VEM O PROGRAMA
)
```

```
10 LET p$=""
20 PRINT "DIGITE A SENHA"
30 PAUSE 0
40 LET k$=INKEY$: IF k$=""
THEN  GOTO 40
50 LET p$=p$+k$
60 IF LEN p$<>7 THEN  GOTO 30
70 IF p$<>"bananas" THEN
STOP
80 PRINT "O.K."
90 REM (Em seguida vem o rest
o do programa"
```

Modifique a linha 20 do programa anterior para:

```
20 GET K$:IF K$=""THEN 20
```

Essa rotina começa com um cordão vazio (**P$**), ao qual vão sendo progressivamente acrescentados os caracteres, um a um, até que a extensão da senha esteja correta. Como os caracteres entrados não são impressos na tela, o programa impede que as pessoas olhem o que está na tela e aprendam a senha.

O **INKEY$** (e outros comandos similares) é igualmente empregado para "congelar" um programa. Isto é útil quando uma tela repleta de informações deve ser examinada. Após a parte do programa que imprime a informação, coloca-se um comando **INKEY$** ou **GET** no programa de modo a bloquear a listagem antes de limpar a tela.

COMPUTADORES QUE NÃO TÊM O INKEY$

Infelizmente, o interpretador Applesoft BASIC, dos micros da linha Apple II, não dispõe de uma função tão poderosa como o **INKEY$**, para efetuar "varreduras" no teclado. Existe apenas o comando **GET$**, mas este bloqueia o andamento do programa toda vez que é encontrado por ele. Dessa forma, se quisermos, por exemplo, fazer programas de jogos em que um míssil ou um disco voador precisa continuar se movendo pela tela enquanto o jogador não pressiona a tecla que comanda um disparo, é necessário "imitar" a ação do **INKEY$**, executando o pequeno programa a seguir (não tente entendê-lo ainda; estudaremos os comandos **PEEK** e **POKE** numa lição posterior):

```
100 LET K$=""
110 LET K=PEEK(-16384)
120 IF K<128 THEN RETURN
130 LET K$=CHR$(K-128)
140 POKE -16368,0
150 RETURN
```

Esta sub-rotina, quando chamada (por meio de um **GOSUB 100**), retornará, através da variável **K$**, o caractere pressionado no teclado. Se **K$** retornar com valor nulo, isso significa que nenhuma tecla foi pressionada. Por isso, podemos colocar a chamada à sub-rotina em um laço, assim:

```
30 GOSUB 100
35 IF K$="" THEN GOTO 30
```

O comando **PEEK** examina a locação de memória -16384, que guarda um código numérico. Se esse número for igual a 128 ou maior, significa que uma tecla foi pressionada e que o seu código ASCII será deduzido pela expressão da linha 130 e convertido para o caractere armazenado em **K$**. O comando **POKE** zera uma outra locação da memória, para permitir uma nova varredura.

O interpretador BASIC do TK-2000 não dispõe de uma função tão poderosa como o **INKEY$** para efetuar "varreduras" no teclado. Existe apenas o comando GET (de funcionamento idêntico ao do Apple II), que tem a desvantagem de bloquear o andamento do programa, sempre que é encontrado por ele. Assim, para programarmos jogos em que um míssil ou um disco voador, por exemplo, precisa continuar se movendo pela tela enquanto o jogador não pressiona a tecla que comanda um disparo, é necessário "imitar" a ação do **INKEY$** através de uma rotina diferente da que foi apresentada para os micros da linha Apple II.

A locação 39 da memória RAM do TK-2000 retém um valor numérico correspondente à última tecla pressionada. Assim, a sub-rotina abaixo retorna esse código toda vez que uma tecla for pressionada:

```
100 LET K=0
110 LET K=PEEK(39)
120 IF K=48 THEN GOTO 100
130 RETURN
```

A locação de memória 39, no TK-2000, contém sempre o número 48, enquanto nenhuma tecla for pressionada.

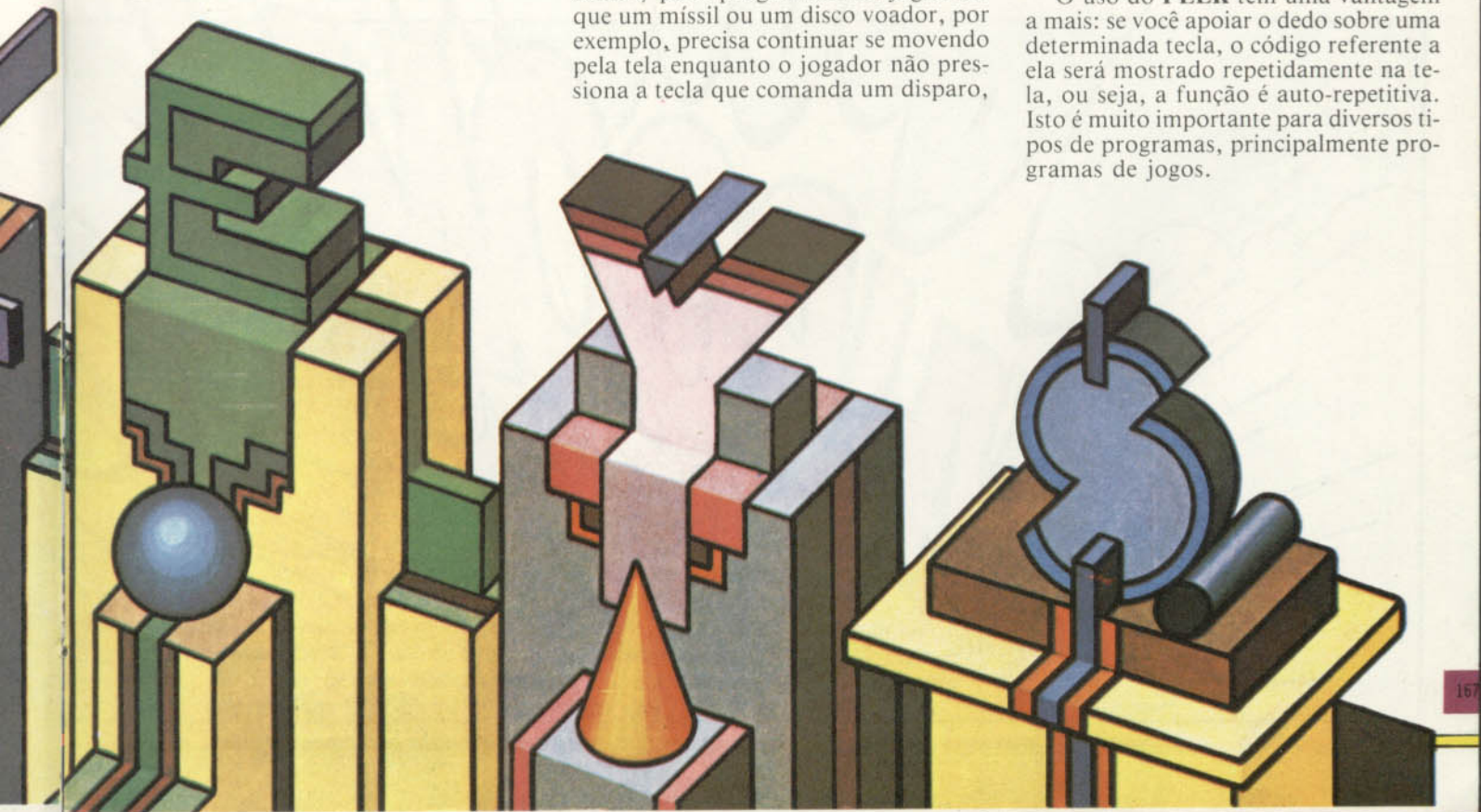
Assim, é fácil incluir uma chamada a essa rotina (**GOSUB 100**) no ponto do programa em que se torna necessário verificar se alguma tecla foi pressionada e, em caso positivo, qual foi essa tecla.

Para descobrir o código gerado por cada tecla do TK-2000, rode o pequeno programa abaixo e pressione sucessivamente as teclas que quer descobrir. O código correspondente será mostrado na tela. Pressione **<CTRL><C>** para interromper o programa:

```
10 HOME
20 LET K=PEEK(39)
30 IF K<>48 THEN PRINT K
40 GOTO 20
```

O uso do **PEEK** tem uma vantagem a mais: se você apoiar o dedo sobre uma determinada tecla, o código referente a ela será mostrado repetidamente na tela, ou seja, a função é auto-repetitiva. Isto é muito importante para diversos tipos de programas, principalmente programas de jogos.

QUEBRE A BARREIRA DO SOM

Bipes, explosões, ruídos extraterrenos: seja qual for a sua escolha, todos esses sons podem ser produzidos por programas simples em BASIC, tornando seus jogos mais eletrizantes.

Os programas de jogos de ação incluem, normalmente, diversos tipos de efeitos sonoros — explosões, tiros, zumbidos, pequenas melodias e outros ruídos — que os tornam mais excitantes e atraentes.

Esta lição tem por finalidade proporcionar a você uma pequena "biblioteca" de sons adequados a programas de jogos. Você pode utilizá-los tal como estão ou desenvolvê-los livremente, como parte de conjuntos sonoros mais complexos.

Lembre-se de que não existem regras sumárias ou definitivas para se produzir efeitos sonoros. Se o seu jogo precisa, por exemplo, de um ruído semelhante ao de alguém mergulhando numa piscina, você terá que sentar-se em frente ao computador e experimentar várias vezes, até conseguir um efeito convincente. Por outro lado, alguns sons aparentemente improváveis poderão ser bem aproveitados, caso você consiga inventar os tipos adequados de gráficos.

A incorporação de efeitos sonoros em seus programas depende da complexidade destes e da freqüência com que tais efeitos serão utilizados. Assim, ruídos simples que entram apenas uma vez em um programa, devem ser colocados após uma declaração **IF... THEN**. Efeitos repetitivos ou mais complexos, por sua vez, precisam de uma sub-rotina.

O grau de sofisticação dos efeitos sonoros depende não só da habilidade do programador, mas também dos recursos de programação de sons de cada computador. Desse modo, o gerador de som dos micros compatíveis com o Sinclair Spectrum (por exemplo, o TK-90X) é limitado a um simples tom puro (chamado de *bipe*, em jargão computacional) obtido por meio do comando **BEEP**, e cuja freqüência e duração podem ser controlados pelo usuário. Um ruído semelhante pode ser produzido, no microcomputador brasileiro TK-2000, mediante o comando **SOUND**. Em contrapartida, os micros das linhas TRS-Color e MSX possuem um gerador de sons bastante sofisticado, capaz de sintetizar uma série imensa de efeitos diferentes. Esse gerador dispõe de vários comandos específicos na linguagem BASIC. O Apple II, o ZX-81 e o TRS-80 só permitem efeitos sonoros satisfatórios através de programas especiais em linguagem de máquina; por isso não serão tratados nesta lição.

O Sinclair Spectrum conta, como já foi assinalado, com um único recurso para a programação de efeitos sonoros em BASIC: o comando **BEEP**. Simples de programar, esse comando pode ser usado para "envenenar" consideravelmente os seus programas. No TK-2000 temos um comando equivalente: o **SOUND**. Para exemplificar, veja a seguir uma rotina que produzirá uma série de sons idênticos:

- INVENTE O SOM CERTO
- COMO INCORPORAR EFEITOS SONOROS AO JOGO DE LABIRINTOS ALEATÓRIOS
- APRENDA A USAR **BEEP**, **PLAY** E **SOUND**
- EXPLOSÕES, TIROS, DESTRUIÇÃO
- DO RUÍDO ÀS NOTAS MUSICAIS MAIS EXPLOSÕES

```
8000 FOR n=1 TO 12
8010 BEEP .03,30
8020 NEXT n
```

Como você pode ver na linha 8010 do programa acima, o **BEEP** é seguido por dois números separados por uma vírgula. O primeiro número determina a duração da nota a ser tocada: quanto maior ele for, mais tempo durará a nota. O valor 1 corresponde à duração de um segundo. Números maiores ou menores do que esse (no caso, frações decimais) funcionarão proporcionalmente.

O segundo número estabelece a freqüência da nota, com o número 0 correspondendo ao dó médio no teclado de um piano. Cada número inteiro acima ou abaixo disso representa um semitom mais alto ou mais baixo — ou seja, é correspondente à tecla mais próxima, em um teclado de piano. Para efeitos sonoros em jogos, freqüências muito baixas, da ordem de −32, são utilizadas para imitar explosões e zumbidos.

```
8000 FOR N=1 TO 12
8010 SOUND 90,3
8020 NEXT N
```

O comando **SOUND** no TK-2000 (veja a linha 8010) funciona da mesma maneira, só que agora o primeiro número determina a freqüência do som, e o segundo a sua duração. Não são permitidos números negativos ou fracionários, nem maiores que 255. A correspondência da freqüência com as notas musicais é um pouco mais complicada, e está ilustrada no manual de programação BASIC do TK-2000.

Os comandos **BEEP** e **SOUND** serão explicados com maior riqueza de detalhes numa próxima lição.

Na rotina acima, o laço **FOR...NEXT** executa uma série de doze notas. Já o programa abaixo contém dois laços, com a variável de controle de cada laço estabelecendo a freqüência das notas. Isto resulta em um efeito que poderá lhe ser útil quando, em um de seus jogos, for necessário criar um som para acompanhar a destruição de um extraterrestre.

```
8000 FOR n=4 TO 0 STEP -1
8010 BEEP .01,n
8020 NEXT n
8030 FOR n=1 TO 4
8040 SOUND .01,n
8050 NEXT n
```

```
8000 FOR N=60 TO 0 STEP -10
8010 SOUND N,3
8020 NEXT N
8030 FOR N=0 TO 60 STEP 10
8040 SOUND N,3
8050 NEXT N
```

Eis aqui uma rotina que emprega o laço **FOR...NEXT** de um modo pareci-

do; o ruído criado, porém, será agora um som de "Congratulações". Você poderá utilizá-lo quando o jogador tiver destruído todos os "monstros" ou para celebrar alguma outra vitória.

```
8000 FOR n=10 TO 60 STEP 5
8010 BEEP .01,n
8015 BEEP .01,n-2
8020 NEXT n
```

```
8000 FOR N=70 TO 150 STEP 5
8010 SOUND N,3
8015 SOUND N-2,3
8020 NEXT N
```

Tente fazer experiências mudando os parâmetros do comando **BEEP** (ou **SOUND**) para produzir efeitos sonoros em seus programas de jogos. Procure utilizá-los também dentro de laços e de sub-rotinas.

EFEITOS SONOROS NO TRS-80

O TRS-80 não tem recursos para a produção de efeitos sonoros, mas é possível fazê-lo emitir alguns sons. Para isto, utiliza-se a interface de saída para o gravador cassete, que gera pulsos no espectro audível, sob controle adequado de um software. Basta ligar o fio que vai do computador até o conector MIC ou AUX, em um sistema de amplificação de som.

O software de produção de som funciona da seguinte maneira: o comando **OUT** envia para a porta de saída do gravador (a de número 255) uma seqüência de bytes 0 e 1. O byte 0 causa um pulso negativo na saída, e o byte 1, um pulso positivo. Alternando-se ambos, temos uma onda de som. Para variar a freqüência do som, varia-se a pausa entre o pulso positivo e o negativo. Com essa técnica, é possível produzir sons em BASIC: basta colocar os comandos OUT dentro dos laços de repetição entre meados de um laço de retardo de tempo:

```
10 FOR I=1 TO 1000
20 OUT 255,0
30 FOR J=1 TO 10:NEXT J
40 OUT 255,1
50 NEXT I
```

Sons mais agudos e efeitos como explosões, sirenas, etc., exigem uma rotina em linguagem de máquina, que pode ser chamada a partir de um programa em BASIC pelo comando **USR**.

SONS PARA O LABIRINTO

Imagine o jogo de labirinto aleatório apresentado na última lição com alguns efeitos sonoros. Note que não há uma versão para o TK-2000. Adicione as linhas 365 e 500 ao programa original e modifique a linha 400.

```
365 BEEP .01,10
400 LET vidas=vidas-1: RESTORE
500: FOR f=0 TO 10: READ a,b:
BEEP a,b: NEXT f: IF vidas>0
THEN  GOTO 260
500 DATA .45,0,.3,0,.15,0,.45,
0,.3,3,.15,2,.3,2,.15,0,.3,0,.
15,-1,.45,0
```

Se você rodar o programa agora, descobrirá que acrescentou uma rotina de som simples mas muito eficaz, que executa uma marcha fúnebre sempre que o jogador perder uma "vida". Tente modificar os valores de a e b para criar efeitos diferentes.

Esse exemplo mostra que você deve ser cuidadoso quando utilizar efeitos sonoros em jogos, já que o computador interrompe o que está fazendo durante a execução do **BEEP**, criando assim algumas pausas inoportunas. Mesmo que você inclua apenas efeitos de curta duração, isso poderá tornar o seu jogo muito lento.

O MSX possui um dispositivo sonoro muito sofisticado que emprega três "vozes" altamente controláveis, ou canais sonoros. Esse dispositivo permite uma grande variedade de ruídos, muitos dos quais podem ser incorporados aos seus programas de jogos.

Tais ruídos podem ser criados por meio das "vozes", isoladamente ou em grupos, de modo a formar diversas combinações. Uma explicação mais detalhada sobre programação de efeitos sonoros no MSX será desenvolvida em um artigo posterior de BASIC.

O MSX utiliza um segundo microprocessador interno somente para produzir sons e efeitos sofisticados: algumas linhas de programa são suficientes para isso. Esse microprocessador é composto por 14 registros que funcionam de maneira semelhante às posições de memória do computador. O som é produzido de acordo com o conteúdo de tais registros, ou seja, suas características dependem dos números armazenados nos registros. Os valores adequados são colocados nesses registros por intermédio do comando **SOUND R, N** (**R** é o número do registro, e **N** o número que determinará as características do som).

O MSX conta ainda com uma linguagem "macromusical" disponível mediante o comando **PLAY**. Quando se deseja programar uma melodia (é necessário entender um pouco de música) é mais fácil utilizar esse comando e não se preocupar com tantos registros. O **PLAY** é, contudo, um pouco lento e não produz ruídos, só notas musicais. Para entendê-lo melhor, leia a seção dedicada ao TRS-Color, mais adiante.

JOGO SONORO NUM LABIRINTO

Se você gravou o programa de criação de labirintos aleatórios para o MSX (apresentado na última lição de programação de jogos), certamente achará interessante acrescentar as três linhas que listamos a seguir.

```
1330 IF X<>LX OR Y<>LY THEN LX=
X:LY=Y:PLAY "V15T255O5L64DG"
1340 IF F=1 THEN F=0:SC=SC+(TI-
TIME):TI=TI-10:PLAY "V15T1000O3B
CDEFGA":GOTO 1220
1500 PLAY "V15T255O2L2CR64L4CR6
4L12CR64L2CR64L4D#R64L8DR64L4DR
64L8CR64L4CR64O1L8BR64O2L2C":LI
=LI-1
```

A primeira linha produz som para o movimento do homenzinho; a segunda, para o momento em que ele encontra o tesouro; a terceira toca uma "marcha fúnebre" quando ele perde uma "vida".

Como dissemos, o comando **PLAY** só é adequado para programação de melodias. Quando desejarmos ruídos (sons

não musicais) ou uma velocidade maior, deveremos utilizar o comando **SOUND**.

Muitos valores (até 13) devem ser colocados nos registros adequados, para a emissão de um determinado som. Comece a aprender como fazê-lo, digitando as seguintes linhas:

```
10 SOUND 7,7
20 SOUND 6,20
30 SOUND 8,15
```

Rode o programa e ouvirá um ruído semelhante ao das ondas do mar quebrando na praia (use a imaginação).

Vejamos os registros utilizados. Na linha 10 colocamos o valor 7 no registro 7. Este último determina os canais de som a serem utilizados e se eles produzirão música ou ruído. O que importa no momento é que o valor 7 ativa os três canais para criar ruídos. O valor 56 ativaria os três para sons musicais. Qualquer outro valor entre 1 e 63 utilizaria os canais de uma maneira mista. Na realidade, deve-se converter o valor utilizado para a forma binária de modo a ficarmos sabendo o que cada canal produzirá: música ou ruído.

Na linha 20, o registro 6 determina a freqüência do ruído produzido, podendo conter valores de 0 a 63. Números menores determinam sons mais agudos. Na linha 30, o registro 8 controla o volume do som produzido pelo canal A, podendo conter um número entre 0 e 15. Geralmente, é melhor colocarmos o valor máximo e ajustarmos o volume da TV. Outras vezes, contudo, podemos criar efeitos interessantes variando o volume. Acrescente as seguintes linhas ao programa para obter um som mais próximo do real. Isto é conseguido variando lentamente o volume nas linhas 40 e 70.

```
30 FOR I=10 TO 15 STEP 1E-03
40 SOUND 8,I
50 NEXT
60 FOR I=15 TO 10 STEP -1E-03
70 SOUND 8,I
80 NEXT
90 GOTO 30
```

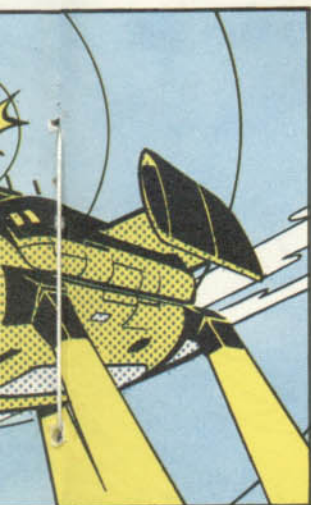
Modifique o valor do **STEP** nas linhas 30 e 60 de -1E-3 para .5 e você ouvirá o barulho de um trem a vapor em movimento. O programa listado a seguir permite que você experimente e modifique vários efeitos sonoros, variando os valores dos diversos registros.

```
5 CLS:R=RND(-TIME)
10 INPUT "Número do programa (1
-13): ";P
15 INPUT "Envoltória (1-fixa ,
2-programável): ";E
16 EE=14+E
20 IF E=1 THEN 35
25 INPUT "Forma da onda sonora
(8-15): ";W
26 SOUND 13,W
30 INPUT "Período do ciclo (0-6
5384): ";F
31 H%=F/256:L%=F-256*H%
32 SOUND 11,L%:SOUND 12,H%
35 INPUT "Música ou ruído (M ou
 R): ";M$
36 IF M$="M" THEN SOUND 7,56:A=
0:T=255:GOTO 40
38 IF M$="R" THEN SOUND 7,7:A=6
:T=63:GOTO 40
39 GOTO 35
40 SOUND 8,EE
60 ON P GOSUB 80,90,100,110,120
,130,140,150,160,170,180,190,20
0,210
70 SOUND 8,0:END
80 SOUND 1,5:FOR Z=1 TO T STEP
.3
85 SOUND A,Z:NEXT:RETURN
90 SOUND 1,0:FOR Z=T TO 1 STEP
-.3
95 SOUND A,Z:NEXT:RETURN
100 SOUND 1,0:FOR Z=1 TO 20 STE
P .1
105 SOUND A,ABS(SIN(Z)*T/2)
107 NEXT:RETURN
110 SOUND 1,1:FOR Z=1 TO 200
115 SOUND A,RND(1)*(T+1):NEXT
117 RETURN
120 IF A=0 THEN A=1:T=15
122 FOR Z=1 TO 50
125 SOUND A,T AND ABS(TAN(Z)+5)
127 NEXT:RETURN
130 SOUND A,T/2:SOUND 1,2
```

A linha 5 limpa a tela e prepara o gerador de números aleatórios. Os diversos **INPUT** das linhas seguintes pedem informações a respeito do efeito desejado. Nas linhas 80 a 207 estão treze diferentes sub-rotinas que geram treze efeitos diversos.

Essas sub-rotinas cuidam principalmente da freqüência dos sons — isto é, se eles são agudos ou graves — e de como essa freqüência vai variar com o tempo. Caso se trate de som musical, é utilizado o canal A, cuja freqüência é estabelecida pelos registros 0 (ajuste fino) e 1 (ajuste grosseiro). Em caso de ruído, a freqüência é determinada pelo registro 6. A variável A contém o valor adequado do registro de freqüência, conforme a opção seja música ou ruído. A variável T varia também conforme essa opção, compatibilizando os valores máximos que cada registro de freqüência comporta.

Conforme o valor do registro 7 (linhas 36 e 38) teremos ruídos ou sons musicais. O valor 7 determina ruído, e o valor 56, música (nos três canais).

Uma das características do dispositivo sonoro do MSX é a possibilidade de programar o formato da onda sonora

ou "envoltória". Isto significa que o volume do som muda ao longo do tempo de acordo com uma curva cujo formato pode ser escolhido por nós (formato de serra, triângulo, pulso, etc). Normalmente, o volume é fixado de acordo com o valor do registro 8 (para o canal A). Se colocarmos nesse registro o valor 16 (linha 40) teremos à nossa disposição diversos tipos de envoltórias (veja o formato delas em seu manual). O registro 13 seleciona a forma da envoltória, e os registros 11 e 12 determinam o período do ciclo. Este último valor corresponde à velocidade com que o volume vai mudar. Valores baixos correspondem a altas velocidades.

Comece então a experimentar. Escolhendo o programa 10, por exemplo, com envoltória fixa e sons musicais, você ouvirá um canto de pássaro. As variações são quase ilimitadas.

Para entender melhor como todos esses valores modificam as características do som produzido, faça o seguinte: primeiro ouça todos os treze programas utilizando envoltórias fixas e sons musicais. Depois comece a ouvi-los com as diferentes envoltórias programáveis, mantendo o mesmo programa para poder comparar. Inicialmente, varie apenas a forma da onda, mantendo o valor 1000 para o período do ciclo. A seguir, varie o período, conservando o resto constante. Depois de experimentar os sons musicais, repita a mesma seqüência para os ruídos.

Existem dois comandos no BASIC do TRS-Color que podem ser usados para produzir efeitos sonoros: o **SOUND** e o **PLAY**. Ambos utilizam o mesmo gerador sonoro, embora não seja possível afetar de maneira significativa a qualidade (timbre) da nota produzida. O **SOUND** controla a freqüência e a duração do som, enquanto o **PLAY** permite que você selecione uma seqüência de notas musicais para executar uma melodia, por exemplo. O comando **SOUND** pode ser utilizado para produzir bipes (simples tons sonoros, de freqüência e duração fixas), como mostramos no programa a seguir. Quando rodá-lo, não se esqueça de ligar o volume na sua TV; do contrário, não ouvirá nada!

```
10 FOR Q=1 TO 5
20 SOUND 200,1
30 NEXT Q
```

A série de bipes pode ser alterada mediante a modificação dos números após o comando **SOUND**. O primeiro número controla a freqüência da nota e pode ter um valor de 1 a 255. A nota mais baixa é produzida por 1 e a mais alta por 255 (para alguém que conhece algo sobre música: o valor 89 corresponde à nota dó da escala do meio de um piano).

O segundo número após o comando **SOUND** regula a duração do tom sonoro. Novamente, a gama possível de valores vai de 1 a 255; o valor 16 fornece cerca de um segundo de duração.

Esses bipes podem ser adequados a jogos especiais ou de fantasia, mas não são de muita utilidade quando o tema do jogo é mais realista. Assim como no caso dos efeitos gráficos simplificados para produzir explosões (flashes) descritos anteriormente, deve-se ser criterioso ao utilizá-los. Nunca deixe de empregar efeitos sonoros adequados ao "clima" do jogo.

Quando se precisa de um tipo mais sofisticado de efeito sonoro, é recomendável abandonar o comando **SOUND**, cujos recursos são limitados. Ao invés dele, é preferível utilizar o comando **PLAY**, como será explicado mais adiante.

OUTRA VEZ UM LABIRINTO

Se você gravou o programa de criação de labirintos aleatórios para o TRS-Color (dado na última lição de programação de jogos), poderá melhorar muito o jogo, adicionando os efeitos sonoros a seguir.

Substitua a linha 230 no programa do labirinto pela seguinte:

```
230 IF X<>LX OR Y<>LY THEN PUT
(X1,Y1)-(X1+BS-1,Y1+BS-1),B,PSE
T:LX=X:LY=Y:PLAY"T5005DG"
```

Mude a linha 240 para:

```
240 IF F=1 THEN F=0:SC=SC+(TI-T
IMER):TI=TI-10:PLAY"T1003BCDEFG
A"GOTO 130
```

e substitua a linha 500 por:

```
500 CLS:SCREEN 0,0:PLAY"T302L2C
L4CL12CL2CL4D#L8DL4DL8CL4C01L8B
02L2C":LI=LI-1
```

Agora, rode o programa e escute o que acontece quando o homenzinho se move pelo labirinto. Você obterá um som diferente no momento em que o tesouro for encontrado, e uma "marcha fúnebre" quando o jogador perder uma "vida".

Ao lidar com efeitos sonoros, procure evitar que eles retardem o seu jogo, pois toda vez que produz um som o computador interrompe qualquer outra coisa que esteja fazendo. Em programação de jogos, isso significa que qualquer

movimento na tela cessa enquanto o som estiver sendo produzido. Na verdade, você pode utilizar sons em vez de laços **FOR...NEXT** para introduzir pausas mais ou menos longas nos seus programas. O som na linha 230, por exemplo, é muito curto, pois o espaço de tempo foi reduzido ao mínimo. Trabalhando com sons, você pode até mesmo fazer o ajuste fino de seus programas.

O COMANDO PLAY

O comando **PLAY** opera sempre com base em um cordão alfanumérico, que transporta instruções para o computador. O cordão na linha 230 do programa anterior contém as instruções **T50O5DG**. O **T** significa Tempo (ou velocidade, se você preferir) e pode ser estabelecido em qualquer valor de 1 a 255. A letra **O** é a oitava, e pode assumir um valor de 1 a 5 (1 é o valor mais baixo, e 5 é o mais alto). As duas últimas letras, **D** e **G**, são notas, no sistema americano de notação (C é dó, D é ré, E é mi, etc.). De um modo muito geral, o que **T50O5DG** significa é: ''toque duas notas altas rapidamente''.

Tente alterar os valores de **T** e **O** para ver que tipos de efeitos sonoros você poderá obter.

Na linha 240, o cordão é **T10O3BCD EFGA**, que executa uma escala crescente.

A marcha fúnebre na linha 500 é mais complexa. Uma explicação detalhada de como escrever música para o computador deverá esperar mais um pouco, mas observe o quanto a duração da nota L é variada durante a melodia. Na verdade, você pode variar qualquer um dos parâmetros em um cordão, em qualquer ponto durante a melodia, tornando assim o comando **PLAY** muito flexível.

EXPLOSÕES

O comando **PLAY** pode igualmente ser utilizado para produzir efeitos sonoros simulando explosões, que serão de grande utilidade na maioria de seus jogos de ação. Os seguintes efeitos sonoros podem ser utilizados sozinhos, ou com as imagens da lição ''Bombardeios e Explosões'' (páginas 121 a 127).

Tente esse efeito:

```
10 PLAY"T160O1L30GF#FE#D#DC#D#D
FE#"
```

Você poderá usá-lo quando algo for atingido ou explodir na tela. Incorpore-o, por exemplo, ao programa da página 64, acrescentando o comando **PLAY** à linha 270, antes do comando **GOTO**.

Um interessante som de reverberação pode ser conseguido assim:

```
10 PLAY"T80O1L2BFBFBFBFBFBFBFBF
BF"
```

Na oitava mais baixa, duas notas se repetem sem cessar. Tente manipular a velocidade e a duração das notas para obter variações sobre esse som.

Você poderá conseguir uma repetição de seus efeitos sonoros utilizando uma rotina como a que se segue:

```
10 F$="T10O02L10AGBE#DFADF#"
20 F$=F$+F$
30 PLAY F$
```

A linha 10 contém o cordão de instruções para o som. A linha 20 concatena o cordão, de modo que, quando a linha 30 o executa, o som se repete.

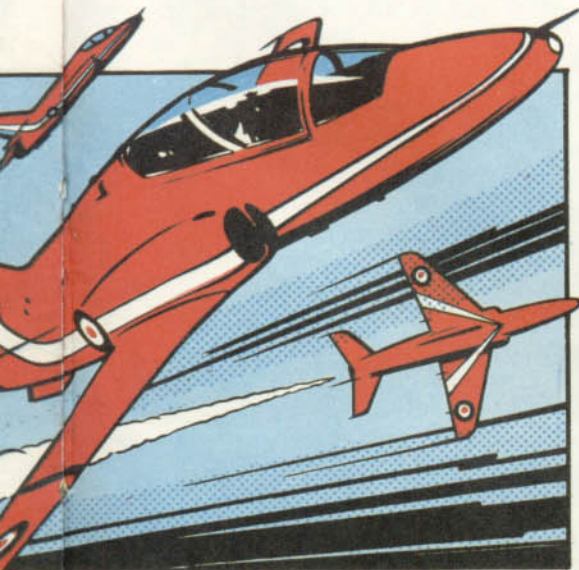
UMA SIRENE

A repetição de um som tem outras aplicações. Se você deseja obter o ruído de uma sirene, empregue um programa como esse:

```
10 CLEAR 250
20 F$="T150O4L8CDD#EF#FGF#FED#D
C#CD#DC#C"
30 F$=F$+F$+F$+F$
40 PLAY F$
```

Para se obter o efeito de sirene, constrói-se um longo cordão, concatenando-se quatro partes do **F$** original, antes de executá-lo. O computador TRS-Color oferece, ao ser ligado, uma quantidade limitada de espaço para variáveis alfanuméricas, de modo que o cordão que contém o novo efeito sonoro será muito maior do que a memória disponível.

Portanto, a linha 10 é usada para reservar mais espaços para variáveis alfanuméricas, por intermédio do comando **CLEAR 250**, que estipula 250 bytes para esse fim. Tenha cuidado em reservar os 250 bytes extras de memória quando você utilizar o efeito sonoro em um programa de jogos; caso contrário, você obterá uma mensagem de erro: OS — *out of string space*.

Não fique desapontado se os seus experimentos não produzirem sons comparáveis aos melhores efeitos em jogos comerciais. Alguns efeitos muito bons são possíveis em BASIC, mas os mais sofisticados são escritos em código de máquina — um tópico que será abordado futuramente em INPUT.

MEMÓRIAS SÃO FEITAS ASSIM

Dividida em RAM e ROM, a memória do computador armazena dados, resultados e programas. Quase tão complexa quanto a memória humana, dela depende o funcionamento da máquina.

A memória dos micros é constituída por milhares de minúsculos circuitos integrados de silício que podem ser ligados e desligados individualmente. Esses circuitos são organizados, dentro do chip, em grupos de oito. Cada grupo representa um byte, isto é, um número binário de oito bits, ou o correspondente a dois dígitos em hexadecimal. Internamente, tal número é usado para definir um endereço para cada locação de memória.

O ESPAÇO DE MEMÓRIA

Em computação, um K é aproximadamente análogo ao k (quilo) do sistema métrico. Na realidade, 1K em computação equivale ao número hexadecimal 400, que é igual a 1 024 em decimal. Assim, se quisermos identificar 64K de memória, devemos numerá-los de 1 a 65 536 em decimal (ou, de 0000 a FFFF, em hexa).

O número hexadecimal de quatro dígitos atribuído a uma locação de memória é conhecido como o seu endereço.

Os micros aqui citados podem atingir uma memória interna de 64K, pois contam com um espaço de endereçamento de dezesseis bits. Entretanto, não é esta a quantidade habitual de memória anunciada pelos fabricantes. O modelo mais comum do Spectrum (por exemplo, o TK-90X, no Brasil) tem 48K; o TRS-Color, 32K; e o Apple II e o TRS-80, 48K cada um. De fato, porém, todos eles têm 64K de espaço de memória, ao todo. Esse número refere-se apenas à quantidade de memória que o usuário ou o programador podem utilizar: os outros 32 ou 16K, conforme o caso, são reservados para utilização pela própria máquina. Nos micros citados, os fabricantes numeram as locações de memória de 0000 a FFFF. O Spectrum ou o ZX-81 de 16K (os menores) — que têm na realidade 32K de memória ao todo — numeram suas locações de memória de 0000 a 7FFF.

ROM E RAM

Além da organização básica em bits e bytes, o espaço de memória do computador é dividido em subunidades: as *páginas,* definidas como *conjuntos endereçáveis* de 100 locações de memória em hexa, cada uma (256 em decimal).

A *página 0 (zero)* vai de 0000 a 00FF; a *página 1 (um),* de 0100 a 01FF, e assim por diante. A organização em páginas facilita a programação do sistema, pois páginas específicas podem ser reconhecidas pelo hardware da UCP.

Existem dois tipos de memória interna: ROM e RAM. ROM (*Read-Onlu Memory*) significa *memória somente de leitura.* Ou seja, pode-se ler o conteúdo das locações de memória ROM mas não se pode escrever nelas. Protegida contra alterações externas, a informação contida é fixada permanentemente quando os chips ROM são produzidos.

A memória ROM guarda as instruções operacionais e o interpretador que traduz seus programas BASIC para código de máquina. Os códigos contidos na ROM também são responsáveis pela imposição de uma estrutura ao resto da memória. Aberta ao usuário, a RAM (*Random Access Memory*) significa *memória de acesso aleatório.* Algumas locações na RAM são controladas pela ROM: se você tentar escrever nessas lo-

cações reservadas de memória, os programas gravados na ROM se encarregarão de fazê-las voltar ao seu conteúdo original. Apesar disso, RAM é uma espécie de folha em branco, na qual você pode escrever o que quiser.

PONTEIROS E INDICADORES

Os mapas de memória mostrados adiante são uma representação gráfica da memória RAM e ROM de cada tipo de micro. As áreas representadas, contudo, não estão em sua posição física exata, pois o espaço da memória é dividido em diferentes chips. Alguns dos limites entre as seções da memória — como a fronteira entre a ROM e a RAM — coincidem com a mudança de um chip para outro. Outros, porém, são flexíveis e suas posições são indicadas por um *ponteiro* ou *indicador* na área de variáveis do sistema.

Um ponteiro é uma locação da memória (ou melhor, um par de locações da memória) que armazena o endereço de uma outra locação — neste caso, o início de uma seção particular da memória. O endereço de qualquer byte de memória é constituído por um número binário de dois bytes de comprimento (dezesseis bits), de modo que precisa ser armazenado em duas locações adjacentes de memória.

S

A memória ROM de 16K do Spectrum vai de 0000 a FFFF e contém o interpretador e o editor de programas em BASIC, bem como várias rotinas de entrada e de saída e o conjunto de caracteres que guardam os dados para as letras do alfabeto, números e outros símbolos gráficos. Os 48K restantes — que vão de 4000 a FFFF — ou os 16K restantes — que vão de 4000 a FFFF — são memória RAM.

As funções das áreas em que a memória RAM está dividida são as seguintes:

A *área de exibição* controla o que é mostrado no vídeo. Cada locação da memória corresponde a uma linha de oito pixels.

A *área de atributos* controla as cores de fundo **(PAPER)** e de frente **(INK)** das 768 locações de caracteres na tela, e o tipo de exibição a ser feita (intensidade normal, brilhante ou piscante).

A *memória intermediária da impressora* retém a próxima linha do texto que será enviada para a impressora.

A *área de variáveis do sistema* contém as locações que guardam os endereços do início das áreas especificadas acima (apontadores).

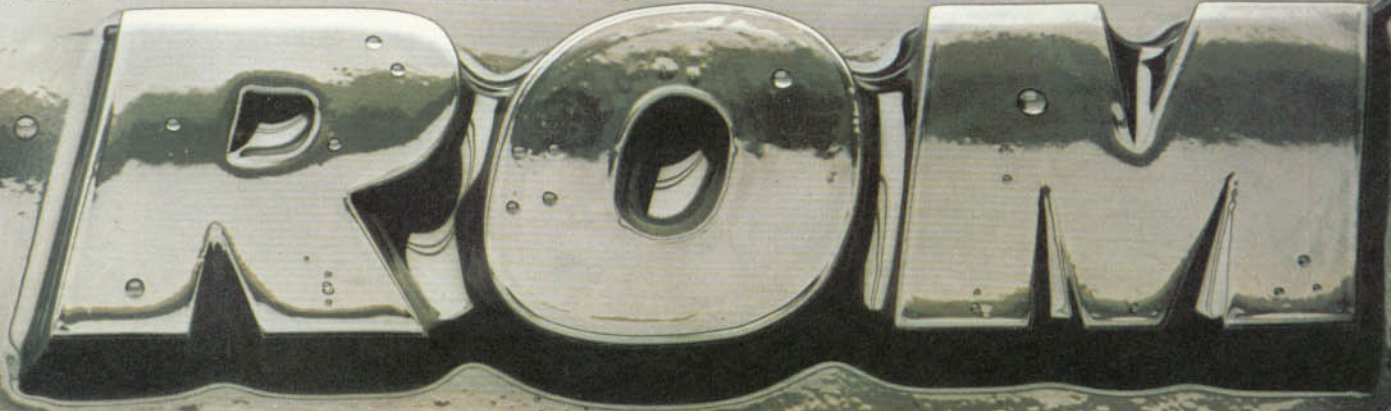
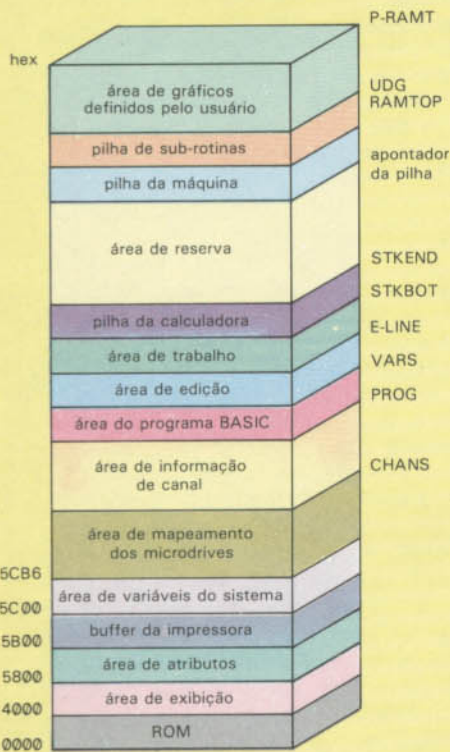
A *área de mapeamento de microdrives* existe apenas nas unidades de microfita (*microdrives*) para o Spectrum, conectadas ao computador. Em caso contrário, o apontador *CHANS*, cujo endereço é armazenado nas locações 23 631 e 23 632 (5C4F e 5C50, em hexadecimal), é deslocado para 5BC6.

A *área de informação de canal* transporta os dados de entrada e de saída. Ela conduz a entrada do teclado para a parte mais baixa da tela de TV e a saída do programa para o restante da tela, para o espaço de trabalho mais acima na memória e para uma impressora.

A *área do programa BASIC* retém as linhas de qualquer programa BASIC digitado ou carregado; seu tamanho depende da extensão do programa. Inicia-se no endereço fornecido pela variável de sistema **PROG,** que é armazenada nas locações 23 635 e 23 636 (5C53 e 5C84) na área das variáveis do sistema. Isto apontará para a locação 23 755 (5CCB em hexa), se nenhum microdrive estiver conectado.

A *área das variáveis* armazena os valores das variáveis que estão sendo utilizadas no programa BASIC em curso. Começa na locação apontada por **VARS,** armazenado nas locações 23 627 e 23 628 (5C4B e 5C4C em hexa) da *área das variáveis do sistema.* Quando um programa é rodado, o início da área das variáveis permanece onde está, mas o final vai crescendo para cima, à medida que novas variáveis são definidas pelo programa em BASIC.

A *área de edição* é o local onde é fei-

ta a edição das linhas BASIC. Quando o <**ENTER**> é apertado, as linhas são copiadas para a área do programa BASIC. A área de edição começa na **E-LINE**, cujo endereço é retido nas locações 23 641 e 23 642 (5C59 e 5C5A em hexa) na área das variáveis do sistema.

O *espaço de trabalho* serve para tarefas gerais, como a armazenagem dos dados de entrada e a concatenação de strings. O **WORKSP** é armazenado em 23 649 e 23 650 (5C61 e 5C62 em hexa) na área das variáveis do sistema.

A *pilha de cálculo* é utilizada para reter números em ponto flutuante, números inteiros de cinco bytes e o conjunto de parâmetros de cinco bytes. Começa em **STKBOT**, cujo endereço ocupa as locações 23 651 e 23 652 (5C63 e 5C64 em hexa), e termina em **STKEND**, que é encontrado em 23 653 e 23 654 (5C65 e 5C66 em hexa).

A *área de memória de reserva* permite que as áreas da memória situadas acima e abaixo dela cresçam até que **STKEND** encontre o indicador de pilha.

Acima dos bytes sobressalentes está a *pilha da máquina,* empregada pela UCP quando um programa BASIC é rodado.

A *pilha de sub-rotinas* (**GOSUB**) armazena os números de linha para os quais um programa deve retornar quando completar cada sub-rotina chamada.

O ponteiro **RAMTOP** indica o final da memória RAM disponível para o armazenamento de programas. Seu endereço é retido em 23 730 e 23 731 (5CB4 e 5CB5 em hexa) na área das variáveis.

Acima de **RAMTOP** há 168 locações de memória que podem ser usadas para armazenar 21 representações ou padrões gráficos definidos pelo usuário. Entretanto, o topo da RAM é armazenado em uma variável do sistema e pode ser deslocado para uma posição mais abaixo na memória, usando-se o espaço de reserva para as pilhas de sub-rotina e de máquina. Isto pode ser feito através de programação em código de máquina. Normalmente, um programa em código de máquina fica acima do **RAMTOP** abaixado.

O apontador **P-RAMT** assinala o *topo físico* da memória RAM; isto é, acima desse ponto não existem mais locações da memória nos chips do Spectrum. O **P-RAMT** é uma variável do sistema, com endereço armazenado nas locações 23 732 e 23 733 (5CB5 e 5CB6 em hexa).

Para verificar o valor de **P-RAMT** ligue a máquina e entre essa linha:

```
PRINT PEEK 23732+256*PEEK 23733
```

Isto retornará o valor 65535 no Spectrum de 48K, ou o valor 32767 no modelo de 16K. Se nenhum destes valores for retornado pela linha, então alguma coisa está errada com o seu computador. Pode-se examinar o conteúdo dos outros apontadores na área de variáveis do sistema, utilizando a mesma linha **PRINT**, apenas substituindo os valores correspondentes às duas locações de memória da variável.

A palavra-chave **PEEK** em BASIC "olha" os conteúdos dos bytes da memória e retorna o equivalente decimal do número que encontrar. O endereço tem o comprimento de dois bytes (requer duas locações da memória para ser armazenado). O Spectrum divide o endereço hexa de quatro dígitos — por exemplo, o endereço normal FF57 de **RAMTOP** — em duas partes de F e 57.

O byte mais baixo, 57, vai dentro do endereço menor, e o byte mais alto, FF, vai no endereço maior. Isto explica por que a segunda das locações da

memória no programa acima é multiplicada por 256; do contrário, o Spectrum poderia retornar ao equivalente decimal de FF, ao invés de FF00.

S

Os micros compatíveis com o modelo Sinclair ZX-81 têm 8K de ROM, com endereços que vão de 0000 a 1FFF em hexa, ou de 0 a 8 190 em decimal. Os outros 8K de memória, de 2000 a 3FFF, não são utilizados.

O mapa de memória mostrado aqui serve para um ZX-81 com um cartucho de expansão de 16K RAM, cujos endereços vão de 4000 a 7FFF em hexa, ou 16 384 a 32 767 em decimal. No caso de o computador ter um cartucho de RAM de 8K ou um de 4K, basta "achatar" as áreas de memória no espaço correspondente de 8K ou 4K, e fazer com que o topo físico da memória RAM passe a ser 5FFF (24 575 em decimal) para o cartucho de 8K, ou 4FFF (20 479 em decimal), para o cartucho de 4K.

Em todos esses casos, as *variáveis do sistema* ocupam a área de 4000 a 4087 em hexa, ou 16 384 a 16 509 em decimal. As outras áreas não são fixas, e os endereços correspondentes aos seus limites — **D-FILE, WARS, E-LINE, STKBOT, STKEND, ERR-SP** e **RAMTOP** — são armazenados como apontadores nessa área das variáveis do sistema.

A *área de programa BASIC*, que vai de 16 509 a **D-FILE**, contém o programa em BASIC do usuário. Da locação **D-FILE** até a locação **WARS**, fica o *arquivo de exibição.*

A *área de variáveis* vai se expandindo à medida que um programa em BASIC é executado e novas variáveis são definidas em seu interior. Essa área tem o seu final assinalado pela locação da memória que contém 80, em hexa.

Entre as locações **E-LINE** e **STKBOT** estão a *área de edição* e o *espaço de trabalho.* A linha que está sendo digitada pelo teclado ocupa a área de edição. Seu conteúdo é transferido para a área do programa assim que a tecla <**ENTER**> é pressionada.

Entre a locação **STKEND** e o indicador da pilha de máquina existe uma área de reserva de memória, para onde as áreas acima e abaixo dela podem se expandir. A *pilha de máquina é usada* pela máquina quando um programa BASIC é rodado; a *pilha de sub-rotinas* (**GOSUB**) armazena os números de linha para os quais um programa deve retornar depois da execução de uma sub-rotina.

O apontador RAMTOP designa o topo físico da memória — isto é, 32 767, 24 575, 20 479 ou 17 407, dependendo do cartucho de expansão RAM conectado à máquina. Tecle a linha seguinte:

```
PRINT PEEK 16388+256*PEEK 16398
```

A linha retorna o valor decimal correspondente à RAM máxima disponível no sistema. Você pode examinar o conteúdo dos outros apontadores na área de variáveis do sistema, utilizando a mesma linha **PRINT** e substituindo os valores correspondentes às duas locações de memória da variável.

A palavra-chave **PEEK** em BASIC "olha" os conteúdos de cada byte da memória e retorna o equivalente decimal do número que encontrar.

O endereço tem o comprimento de dois bytes. O ZX-81 divide o endereço hexa de quatro dígitos — por exemplo, o endereço normal FF57 de RAMTOP — em duas partes de FF e 57. O byte mais baixo, 57, vai no endereço menor, e o byte mais alto, FF, no endereço maior. Isto explica por que a segunda das duas locações da memória no programa acima é multiplicada por 256.

T

O TRS-Color tem uma memória ROM de 32K, que vai da locação &H8000 à &HFFFF. Essa área é dividida em duas outras: uma que controla a entrada e a saída do computador em relação ao cartucho de ROM removível, e outra, não removível, que contém o sistema operacional e o interpretador BASIC.

Os programas em BASIC e suas variáveis são armazenados na memória RAM, entre as locações &H3600 a &H7FFF. Existe também uma *pilha* que ocupa a parte mais elevada de RAM, logo abaixo da ROM, de &H7FFF para baixo. Mas a pilha pode descer um pouco mais, de modo a criar espaço para um programa em código de máquina. Isto é feito alterando-se o valor da variável do sistema, que aponta para a base da pilha.

Uma parte da RAM é reservada para oito *páginas de gráficos* de alta resolução, que não correspondem exatamente às páginas de memória mencionadas antes. Estas contêm 256 locações, ou 0.25K. As páginas de gráficos, ao contrário, têm 1.5K, e cada uma guarda informação gráfica suficiente para preencher toda a tela na modalidade de resolução mais baixa — **PMODE 0** —, enquanto a modalidade de resolução mais alta — **PMODE 4** — necessita quatro dessas páginas para preencher totalmente a tela. O computador reserva automaticamente quatro páginas, mas **PCLEAR** pode ser usado para reservar mais.

Quando não estão em uso, essas páginas são deixadas vazias. Mas, se uma memória extra for requisitada, elas poderão ser limpadas até a página 1 do BASIC. Isto permite o acesso a uma memória extra de 10.5K.

A *tela de texto* ocupa 1/2K entre &H400 e &H600. Isto corresponde a uma locação de memória para cada espaço de caracteres na tela de dezesseis linhas de 32 caracteres. Os caracteres, formados por 96 pontos na tela (oito por doze pixels), são gerados por um *chip gerador de vídeo* separado.

As *variáveis do sistema* estão armazenadas na área que vai de &H000 a &H400.

MAPA DE MEMÓRIA DO TRS-COLOR

Decimal	Área	Hexa
65535		FFFF
	Dispositivos E/S + vetores Área de cartuchos ROM	C000
49152		A000
	ROM BASIC 16K	8000
32768	RAM 32 K Área de programa + variáveis	3400
13824	Página vídeo 8 1.5 K	3000
12288	Página vídeo 7 1.5 K	2C00
10752	Página vídeo 6 1.5 K	2800
9216	Página vídeo 5 1.5 K	2400
7680	Página vídeo 4 1.5 K	2000
6144	Página vídeo 3 1.5 K	1C00
4068	Página vídeo 2 1.5 K	1800
3072	Página vídeo 1 1.5 K	1400
1536	Página de texto 0.5 K	1000
1024	Uso do sistema 1 K	0800
0		0400
		0000

São uma coleção de indicadores ou apontadores, que fornecem os endereços do início de diversas áreas na memória e de outras variáveis do sistema.

O primeiro grupo de 256 bytes — de &H00 a &HFF — é conhecido, no TRS-Color, como *página direta*. Qualquer das páginas de memória pode ser designada como página direta, ou seja, a página cujas locações podem ser endereçadas mediante a utilização de um endereço curto de um byte, em vez do endereço normal e maior de dois bytes. Isto é feito definindo-se o *registro da página direta* no microprocessador.

T

O TRS-80 tem um espaço máximo de memória RAM de 48K, quando usado com o BASIC Nível II (máquinas com gravador cassete), ou com o TRSDOS e equivalentes (máquinas com disquete). Quando esses micros são usados com o sistema operacional CP/M, essa memória pode ser expandida internamente para 64K de RAM. Trataremos aqui apenas do mapa de memória de máquinas compatíveis com o modelo III da Radio Shack, como é o caso do Prológica CP-500.

O espaço total de memória interna do TRS-80 é de 65 535, ou HFFFF, em hexa. O espaço que vai do endereço 0 a 14 435 (ou H0000 a H3800) é ocupado pela memória ROM do sistema: 12K de EPROM contendo o sistema operacional e o interpretador BASIC Nível II, e 2K de EPROM adicionais para uso do sistema (total de 14K de ROM).

A memória RAM começa sempre na locação 14 336 e se estende até o topo físico da memória, que pode ser 32 767 (máquinas com 16K), 49 151 (máquinas de 32K) ou 65 535 (máquinas de 48K). Os endereços em hexa são: H7FFF, HBFFF e HFFFF.

A RAM é dividida, por sua vez, em duas partes principais: uma reservada para uso do sistema, que vai de 14 336 a 17 384 (H3800 a H43E8) e outra, que começa em 17 385 (H43E9) e vai até o topo lógico da memória (**RAMTOP**). Essa locação está armazenada na área de variáveis do sistema, e é definida pela primeira vez quando o computador é ligado, e faz a pergunta ao operador (MEM SIZE?).

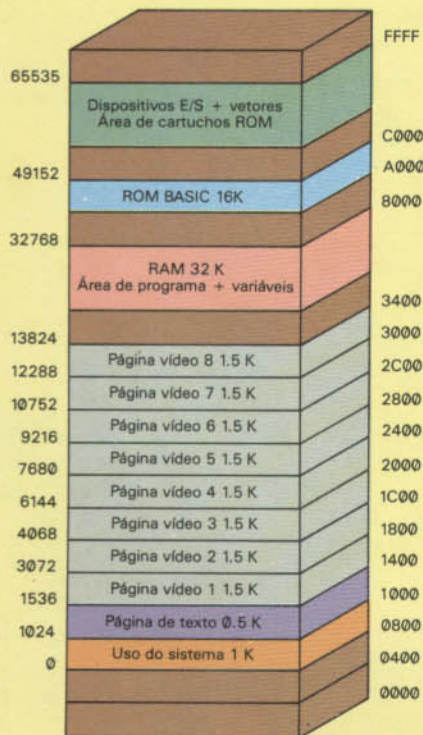
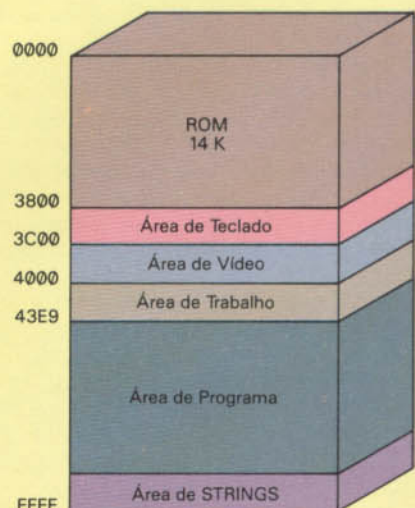
A área do sistema é subdividida, por sua vez, em diversas áreas de comprimento fixo e uma de participação flexível. As áreas de comprimento fixo são as seguintes (veja o gráfico):

A *área matricial do teclado* contém as locações de memória mapeadas no teclado. Qualquer tecla pressionada "liga" bytes correspondentes nessa área (que vai de 14 336 a 15 359).

A *memória de vídeo* também corresponde a um mapeamento direto das 1 024 posições de caracteres na tela, em memória RAM. Estende-se de 15 360 (canto superior esquerdo da tela) a 16 383 (canto inferior direito). Assim, uma posição N na tela (correspondente ao parâmetro do **<PRINT>**) é armazenada na posição absoluta de RAM igual a **15.360 + N.**

A área que vai das locações 16 384 a 17 384 (mil bytes, portanto) é a *área do sistema,* que contém apontadores, *buffers,* indicadores e outras locações especiais de uso particular do interpretador BASIC e que podem ser examinados e alterados pelo usuário por meio de comandos **PEEK** e **POKE.** O conteúdo inicial dos apontadores e indicadores é estabelecido quando a máquina é ligada, ou quando é dado um RESET de iniciação.

A *área de programas e dados* começa na locação 17 385 (H3E9). Tem três diferentes compartimentos: a área do programa, a das variáveis e a de cordões alfanuméricos. Os limites entre essas três subáreas são alterados dinamicamente, em função da execução do programa em BASIC e de alguns de seus comandos. Por exemplo, o comando **CLEAR** fixa o tamanho máximo da área de cordões, logo abaixo do topo lógico da RAM. Ao ser introduzido, o interpretador aloca automaticamente cinqüenta bytes a essa área.

O procedimento usual para colocar um programa em linguagem de máquina do usuário é fixar um limite lógico máximo de RAM, seja através da pergunta inicial, seja através da manipulação do ponteiro que contém o endereço. Isto cria uma área

protegida de memória onde é colocado o programa em linguagem de máquina por meio do monitor residente do TRS-80/III ou de programas monitores e Assembler carregados externamente.

Os micros MSX apresentam algumas diferenças entre si. Por isso, tomaremos como exemplo o micro nacional Sharp HB-8000 (Hot-bit). Este dispõe de um espaço máximo de memória de 128K, dividido em quatro *slots,* ou encaixes disponíveis, quando usado com o BASIC para gravador cassete. Essa memória tem RAM disponível de 64K.

O espaço de memória do Sharp é dividido em dois compartimentos de 65 535 bytes, ou &HFFFF, em hexa. No primeiro, o espaço que vai de 0 a 32 767 (ou &H0000 a &H7FFF) é ocupado pela memória ROM; ele contém o interpretador BASIC e o sistema operacional. O espaço restante, de 32 768 a 65 575 (&H8000 a &HFFFF), é utilizado para endereçamento dos cartuchos removíveis de ROM.

A memória RAM básica está no segundo compartimento. A área que vai de &H0000 a &H7FFF (*área livre*) não é controlada pelo interpretador BASIC.

A segunda área da RAM é controlada pelo interpretador BASIC e é dividida em duas partes principais: a primeira, mais alta e de comprimento fixo, é reservada para uso do sistema e vai de &F380 até o topo físico da RAM. É a *área do sistema* e contém apontadores, *buffers,* indicadores e outras locações de uso particular do interpretador BASIC e que podem ser examinados e alterados pelo usuário (comandos **PEEK** e **POKE**).

A segunda área, mais baixa, contém tudo o que é de uso do programa BASIC armazenado e é subdividida em áreas de comprimento variável.

A *área de blocos de controle dos arquivos* tem como limite máximo o topo lógico da área do usuário. Esse valor é definido como sendo &HF380, mas pode ser modificado para baixo (comando **CLEAR**). Se um valor diferente deste for estipulado pelo **CLEAR,** a área livre entre essa locação máxima e a locação &HF380 fica disponível para abrigar programas em código de máquina. A área dos blocos de controle contém *buffers* e indicadores para transferências de entrada e saída. Seu tamanho é definido internamente ou modificado pelo comando **MAXFILES** do BASIC.

A *área de cordões alfanuméricos* vem logo abaixo e contém os cordões definidos durante a execução do programa. Tem tamanho estabelecido pelo comando **CLEAR.** Este pode conter um segundo argumento, que é a definição do topo lógico da RAM. Por exemplo, **CLEAR 200, &HDFFF** fixa esse valor em &HDFFF.

A seguir vem a *área da pilha,* que tem comprimento fixo e contém os endereços de retorno das sub-rotinas (**GOSUB**) e dos laços **FOR...NEXT.**

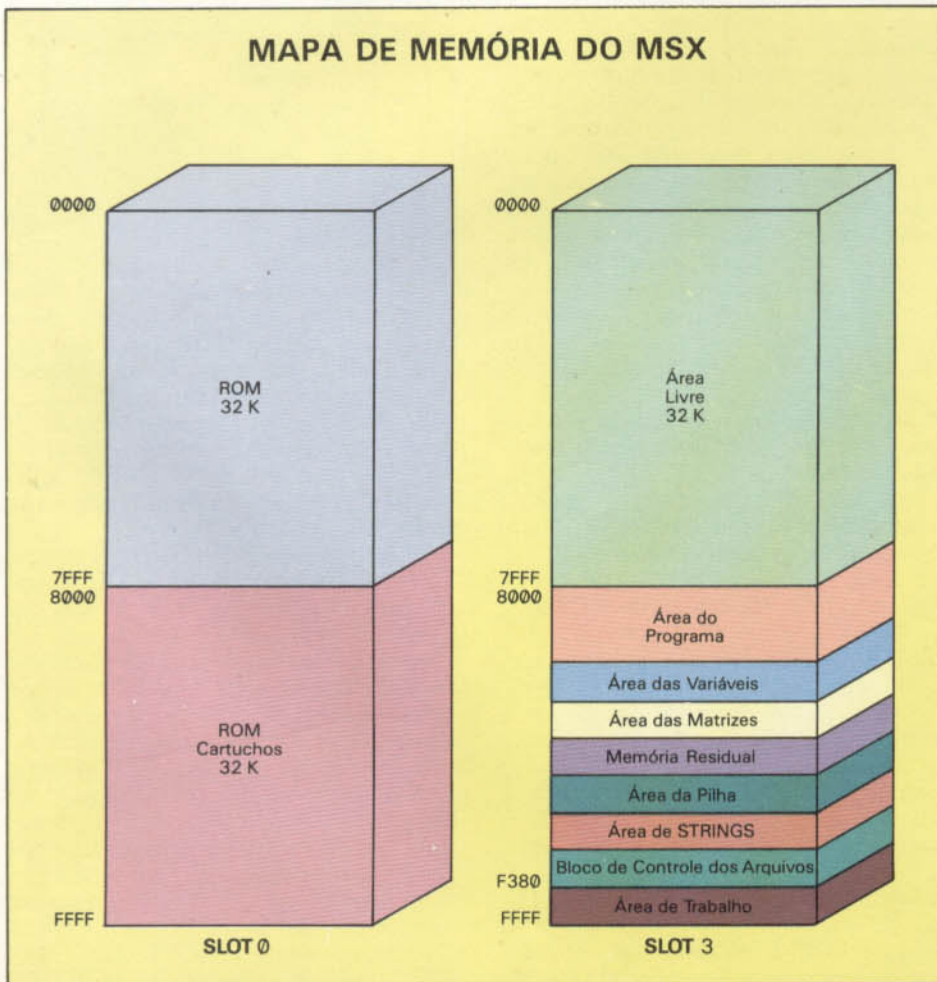
A área que se inicia em 32 768 (&H8000) é a *área de programa,* que abriga o texto do programa. Acima dela existem duas áreas dinâmicas, que crescem à medida que o programa é executado: a *área de variáveis* e a *área das matrizes.* A área de reserva de memória para programa (*área residual*) corresponde à diferença entre a soma das áreas fixas (pilha, cordões e blocos de E/S) e a soma das áreas variáveis (programa, variáveis e texto).

A *memória de vídeo* do MSX também corresponde a um mapeamento direto de posições de caracteres ou de gráficos em diversas páginas. São 16K de RAM que têm seu endereçamento à parte.

A tela de vídeo é tratada pelo microprocessador central como um dispositivo de entrada e saída, e o BASIC dispõe de vários comandos para fazer funcionar esse esquema, inclusive leitura e gravação absoluta (**VPEEK** e **VPOKE**).

MAPA DE MEMÓRIA DO MSX

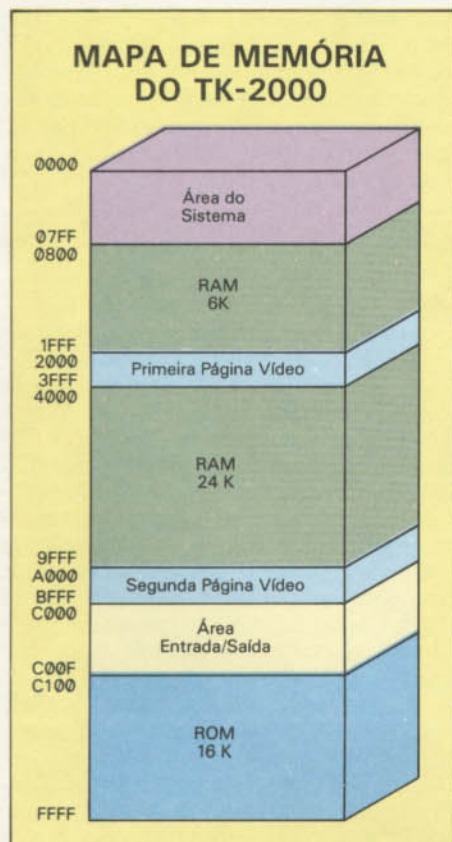
O Apple II e o TK-2000 utilizam o microprocessador 6502, de oito bits. Seu espaço total de endereçamento de memória é de 64K, dividido em 256 páginas de 256 bytes. Essas páginas são alocadas para diferentes funções pela UCP e pelas rotinas operacionais, e constituem o mapa de memória do Apple. Na configuração padrão desses micros, somente 48K são disponíveis para o usuário (RAM), mesmo assim parcialmente, pois uma parte é reservada para uso do sistema. Os 16K restantes correspondem à memória ROM. A forma básica de organização da memória é semelhante para os micros da linha Apple II e para o TK-2000.

A organização da memória do Apple II depende do tipo de linguagem empregada (INTEGER BASIC, FPBASIC ou APPLESOFT), de se está sendo utilizado disquete ou não, e da versão do sistema operacional de disquete (DOS). Além disso, é

possível expandir-se a memória RAM mediante bancos adicionais, para 64K ou 128K. Examinaremos aqui o mapa de memória para a configuração básica, ou seja, um Apple II com 48K de memória RAM e sistema cassete:

A *área da memória ROM,* que se estende das páginas $C1 a $FF (em hexa), contém o código de máquina necessário para a operação e programação da máquina — o interpretador BASIC, o monitor, e, no caso do TK-2000, também um disassembler e um míni-assembler.

A *área de entrada/saída* é uma página de 1K, que inclui uma série de registros, áreas de memória intermediária, apontadores e indicadores relativos aos dispositivos de entrada e saída da máquina (vídeo, som, gravador cassete, teclado, seleção de RAM, etc). Além disso, existem áreas de memória reservadas para as *páginas de vídeo,* que servem para mapear os textos e gráficos que nele aparecem.

Os endereços das áreas de vídeo são diferentes para o TK-2000 e para os micros da linha Apple:

		Apple	TK-2000
TEXTO	Pág. 1	0400-07FF	2000-3FFF
TEXTO	Pág. 2	0800-0BFF	A000-BFFF
HGR	Pág. 1	2000-3FFF	2000-3FFF
HGR	Pág. 2	4000-5FFF	A000-BFFF

O Apple tem quatro áreas de vídeo: duas páginas situadas na parte baixa da memória (antes do início da área de programas) e duas na parte alta da RAM. Já o TK-2000 tem somente duas áreas, ambas na parte alta da memória.

A área de texto do Apple e do TK-2000 também é usada para gráficos de baixa resolução (GR), ao passo que as áreas de gráficos de alta resolução são separadas apenas no Apple.

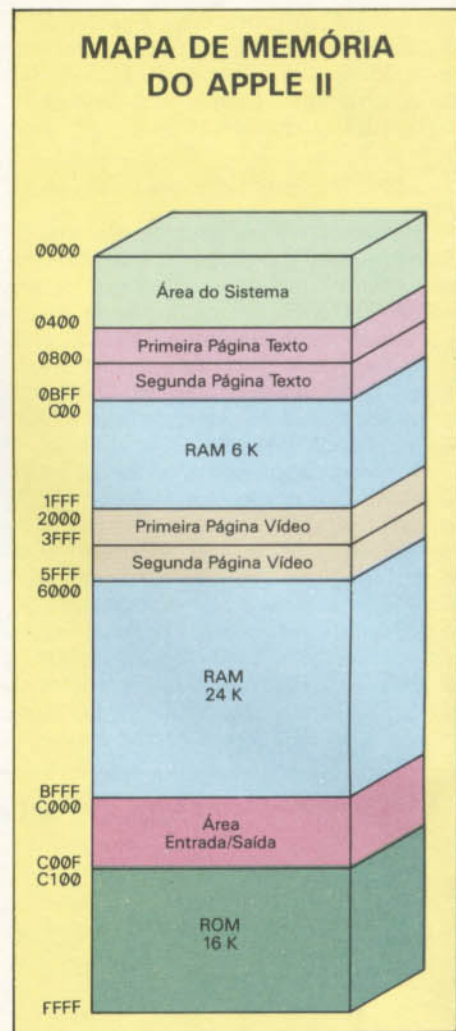
Finalmente, a terceira área é a *área de memória RAM,* que vai da página 0 à página 191 ($00 a $BF, em hexa).

A memória RAM também é subdividida em áreas de tamanho fixo ou variável, conforme a aplicação:

A *área de variáveis do sistema* é armazenada na página 0 e contém apontadores e indicadores atualizados pela UCP. A *pilha da máquina* ocupa a página 1, e tem 256 bytes. Ela serve para diversos fins, podendo inclusive ser utilizada pelo programador em linguagem de máquina; normalmente, ela guarda os endereços de retorno de sub-rotinas e laços em BASIC. A *área de edição,* alocada na página 3, contém o *buffer* (memória intermediária) do teclado, ou seja, a última linha digitada.

A *área de trabalho,* que se estende das páginas 3 a 7, também é reservada para o sistema operacional, para o monitor e para diversas rotinas de E/S. Alguns endereços dessa área podem ser utilizados pelo programador em linguagem de máquina para colocar seus próprios programas, bem como para modificar alguns ponteiros e indicadores que permitem alterar a maneira pela qual a UCP processa os dados na memória e nos dispositivos de E/S.

Finalmente, o espaço que vai da página 8 em diante é a *área do usuário.* Os programas em BASIC começam normalmente na página 8, que também pode ser utilizada para programas em linguagem de máquina. Nos micros da linha Apple, contudo, esse espaço de RAM não fica inteiramente disponível para os programas do usuário. Bem no meio da área do usuário estão as duas páginas de vídeo referidas acima. Usando comandos **POKE,** nos micros da linha Apple, e os comandos **MP** e **MA,** no TK-2000, é possível comutar-se a utilização da primeira área de vídeo, passando-a para a segunda, situada mais acima.

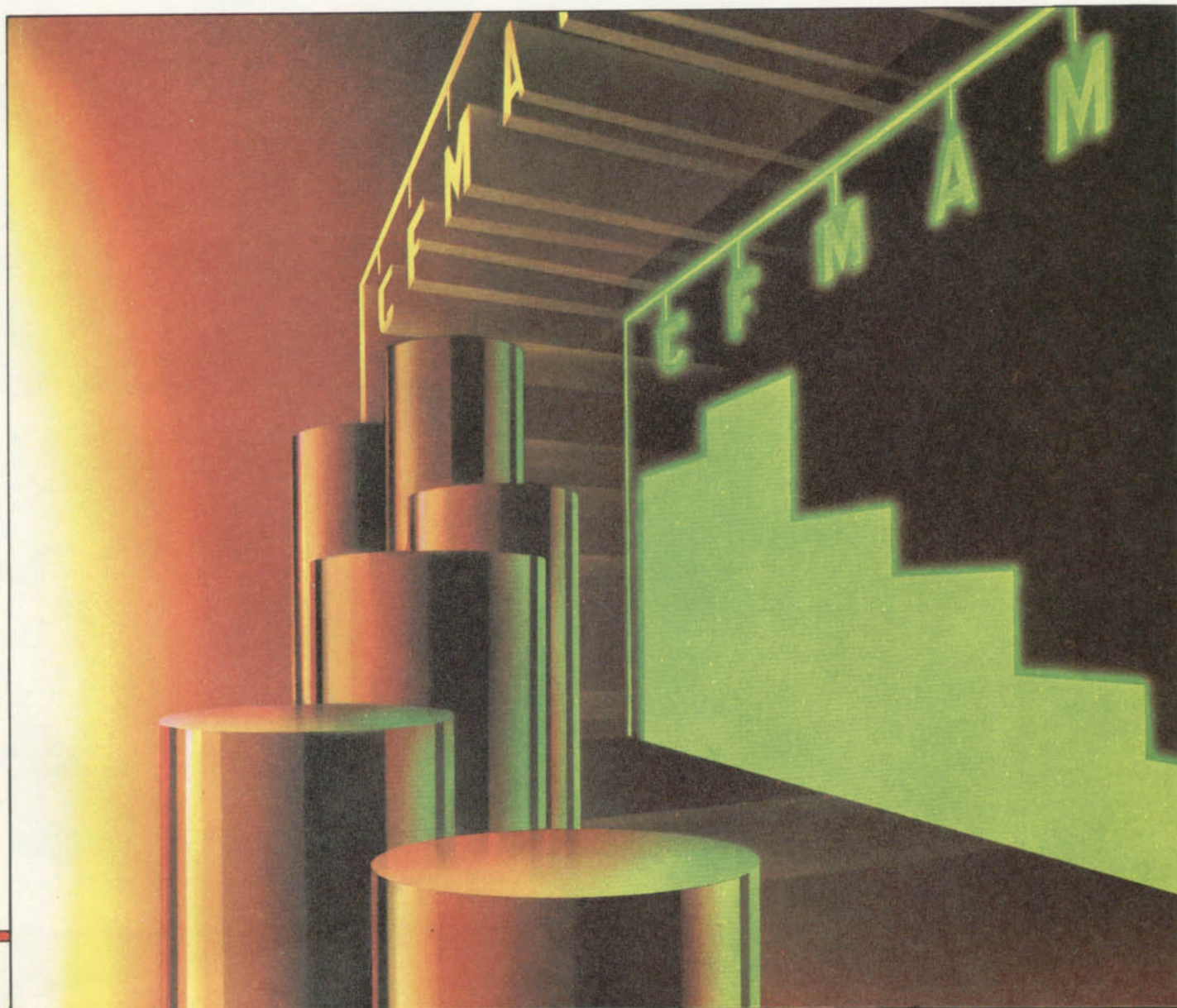
Os programas em BASIC começam a partir de &H0800. Por isso, sobram cerca de seis Kbytes até o começo da primeira área de vídeo. Se você quiser utilizá-la, o truque acima não funcionará. Mas há uma alternativa. A *área de variáveis* do BASIC tem dois limites definidos por dois apontadores na área de variáveis do sistema: **LOMEM,** que define o limite inferior dessa área, e **HIMEM,** seu limite superior. Existem comandos equivalentes em BASIC para alterar esses endereços. Assim, deslocando-se simultaneamente o endereço, de início de um programa BASIC para uma posição *acima* da primeira área de vídeo (isto é feito com dois comandos **POKE**) e os parâmetros **HIMEM** e **LOMEM,** é possível preservar-se a primeira página de vídeo.

O Apple II e o TK-2000 têm a capacidade de selecionar a RAM ou a ROM para execução de programas residentes em memória. Essa opção é armazenada em dois indicadores situados na área de entrada/saída, que podem ser modificados por dois **POKE.** No TK-2000, o apontador **RO** seleciona a ROM e está na locação $C05A. O Apontador **RA** seleciona a RAM e está em $C05B. Ao ser ligada a máquina, é selecionada a ROM, o que ativa o interpretador BASIC residente. O programador pode alterar esses endereços e dar ordem para que o micro execute código de máquina (primeiro endereço da RAM). O botão **<RESET>** e as teclas **<CTRL><RESET>** fazem uma resseleção da ROM.

REÚNA SEUS DADOS EM GRÁFICOS

Digite os números e deixe o micro convertê-los em histogramas coloridos e bem-feitos. Por meio deles, será fácil comparar valores diversos ou verificar suas tendências temporais.

Você já deve ter visto alguns anúncios de programas para aplicações comerciais, nos quais se destaca a elaboração de gráficos sobre estatísticas de vendas, evolução de cotação das ações na Bolsa, etc. Esse tipo de informação, que em geral envolve uma quantidade enorme de números, torna-se de fácil compreensão quando representado por diagramas ou gráficos. E se preparar um gráfico manualmente é trabalhoso, para um computador comercial trata-se de tarefa simples e rápida.

A capacidade de representar informações não é, entretanto, atributo exclusivo dos computadores comerciais. Ela estende-se, também, aos computadores domésticos, constituindo-se em uma de suas áreas de aplicação prática. Todos os computadores apresentados aqui possuem capacidade numérica e gráfica que os habilita a efetuar esse tipo de trabalho, com graus variáveis de precisão e de detalhamento gráfico.

Dificilmente o usuário médio de computadores domésticos precisará lidar com a vasta quantidade de informações geradas, por exemplo, por uma pequena empresa. Mas numerosas questões que lhe interessam diretamente podem ser analisadas de maneira proveitosa em um microcomputador. Por exemplo, a relação entre seus rendimentos e gastos

ao longo de um ano; o peso específico, em períodos distintos, dos diversos itens que compõem seu orçamento; as variações mensais de sua capacidade de poupança.

Além dessas áreas, cujas aplicações se assemelham às comerciais, existem várias outras, sobretudo relacionadas com algum tipo de passatempo, que podem ter seus dados analisados e representados graficamente: a freqüência ao clube local ou a espetáculos teatrais em determinado período, a evolução do número de itens em coleções de livros, discos ou selos e estatísticas de resultados esportivos.

O programa apresentado serve para preparar, rapidamente, um gráfico de barras — ou histograma —, mostrando as variações temporais de algum tipo de valor. Como os eixos do gráfico (horizontal e vertical) ajustam-se automaticamente, pode-se trabalhar com dados semanais, mensais, anuais ou em qualquer outra unidade de tempo.

O intervalo máximo de valores utilizado pelo programa depende do seu computador. Para o TRS-Color, esses valores situam-se entre +99 e −99. Para o Spectrum, entre +1000 e −1000, mais ou menos. Já para o Apple e o MSX, tais valores podem ter qualquer amplitude — unidades, dezenas, centenas, milhares e até milhões, se sua aplicação assim requerer.

ENTRADA DOS DADOS

Ao executar o programa, este apresentará na tela uma lista das opções disponíveis (o menu). Se for selecionada a opção para a entrada de dados, o programa pedirá os títulos dos dois eixos, apresentando-os no momento em que traçar o gráfico.

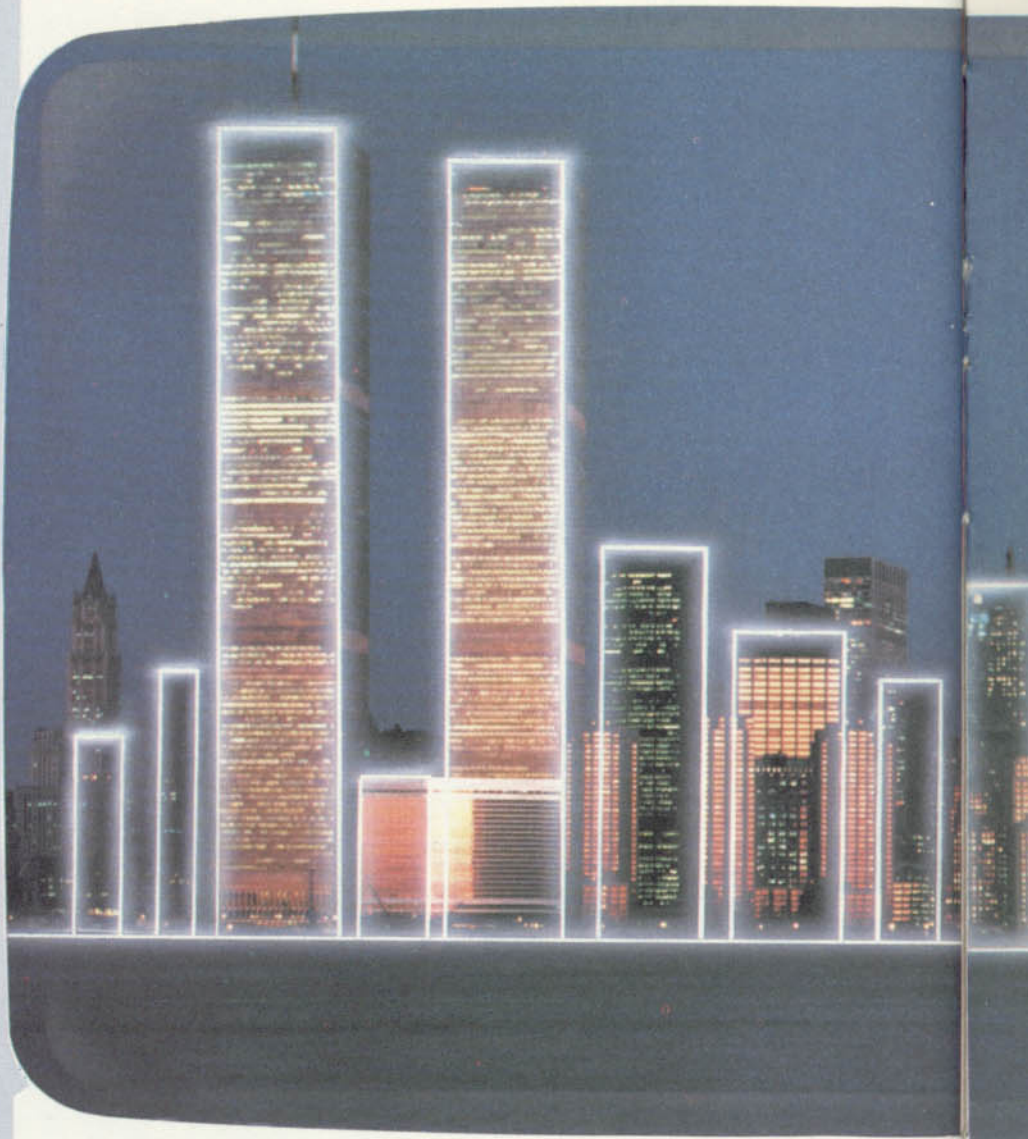
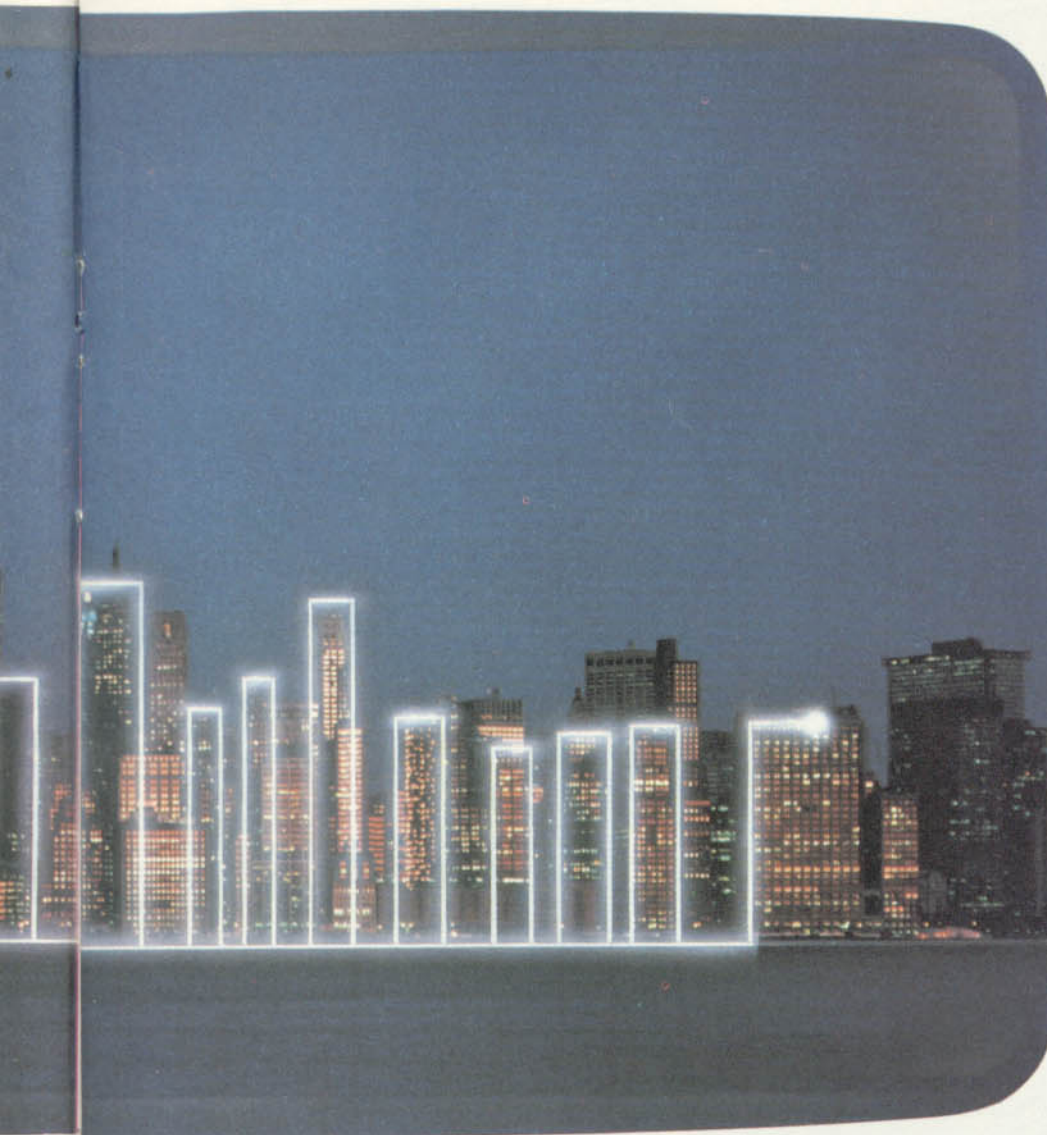
Ao entrar com os títulos dos eixos, tenha sempre o cuidado de digitá-los corretamente e na ordem pedida. O eixo horizontal (X) representa o tempo: semanas, meses, anos ou qualquer outra unidade escolhida. O eixo vertical (Y) representa os valores dados às barras — cruzados, temperatura, número de discos, etc. A altura de cada uma das barras será proporcional a este valor.

Em seguida, o programa pede o número de barras que deve apresentar. O número máximo possível depende, basicamente, da capacidade gráfica do seu computador. O Spectrum, por exemplo, apresenta no máximo 25 barras; no TRS-Color, este número passa a 26; o Apple e o MSX têm capacidade para apresentar até 200 barras, mas seu limite prático é de cerca de 65. Acima deste valor, as barras ficam muito finas ou irregulares, constituindo um quadro visualmente pouco claro.

As perguntas seguintes feitas pelo programa referem-se aos valores dos dados. São apresentados na tela os números seqüenciais das barras e uma pergunta sobre o valor a ser atribuído a cada uma. Esse valor, positivo ou negativo, é, novamente, limitado pela capacidade numérica do seu computador. No TRS-Color, os valores vão de −999 a +999 e, no Spectrum, de −1000 a +1000. O Apple e o MXS, por sua vez, trabalham com valores de qualquer magnitude, já que os gráficos são montados com o uso de escalas.

Se seu computador é um Spectrum ou um TRS-Color, e você deseja representar valores que estão fora da capacidade do programa, a solução é fornecer os dados divididos ou multiplicados por algum fator de conversão (por exemplo, em unidades de centenas, milhares, milhões, etc.).

Quando o último valor for introduzido, surgirá novamente a lista inicial de opções. Você poderá, então, escolher entre modificar algum valor digitado ou partir para a apresentação do gráfico. Se optar pela modificação, os valores que digitou serão apresentados na tela. Siga as instruções para corrigir o valor desejado.

Quando estiver satisfeito com os valores, selecione a opção que permite a apresentação do gráfico. Os usuários do Apple e do MSX possuem apenas uma maneira de apresentar seus gráficos. Usuários do TRS-Color e do Sinclair Spectrum podem escolher entre um gráfico em escala e um gráfico em representação real.

A opção de um gráfico escalonado apresenta o diagrama de barras com as dimensões, ao longo do eixo y, arredondadas para um valor máximo de 10, 100, 1000, etc., dependendo do valor máximo dos dados. Conforme este valor, porém, o gráfico pode não aparecer inteiramente na tela.

A opção de um gráfico em representação real mostra o diagrama de barras ocupando a tela inteira, apresentando os valores reais dos dados (não os valores em escala) ao longo do eixo y.

Para maior clareza, as barras apresentadas pelo Apple, pelo MSX e pelo Spectrum são separadas umas das outras por um pequeno espaço ou uma barra colorida. O TRS-Color apresenta as barras coloridas alternadamente de cinza e amarelo, para os valores positivos, e de vermelho e laranja, para os valores negativos.

COMO FUNCIONA O PROGRAMA

O programa de elaboração de histogramas tem três rotinas principais:

- entrada dos dados que serão exibidos no gráfico;
- edição (modificação) dos dados já entrados;
- elaboração do gráfico.

Os dados são armazenados no conjunto A pela primeira rotina. A capacidade desse conjunto é especificada dinamicamente pelo comando **DIM**, após o usuário indicar o número de dados que deseja utilizar. Quando o programa entra na rotina de elaboração do gráfico, o conjunto de dados é examinado, para que se identifiquem os valores mínimo e máximo nele presentes. Esses valores são armazenados nas variáveis **LO** e **HI**, respectivamente. O valor mínimo predefinido é sempre zero; se os dados apresentarem um valor mínimo maior, este será ignorado pelo programa. O valor máximo é arredondado: 10, 100, 1000, 10 000, etc.

Havendo dados negativos no conjunto A (valor mínimo menor que zero), o programa definirá que o gráfico é de tipo 1; senão, de tipo 2. Nos gráficos de tipo 1, a linha de base (eixo dos x) ficará acima do fundo da tela, e as barras correspondentes aos valores negativos serão dirigidas para baixo.

O escalamento dos valores de um eixo é feito segundo uma regra simples, que leva em conta os valores mínimo e máximo, o número de pixels (pontos gráficos) que cabem no eixo e o arredondamento ou não dos rótulos ao longo do eixo. A fórmula de escalamento é:

$$x' = \frac{x - xmin}{xmax - xmin} \cdot pix$$

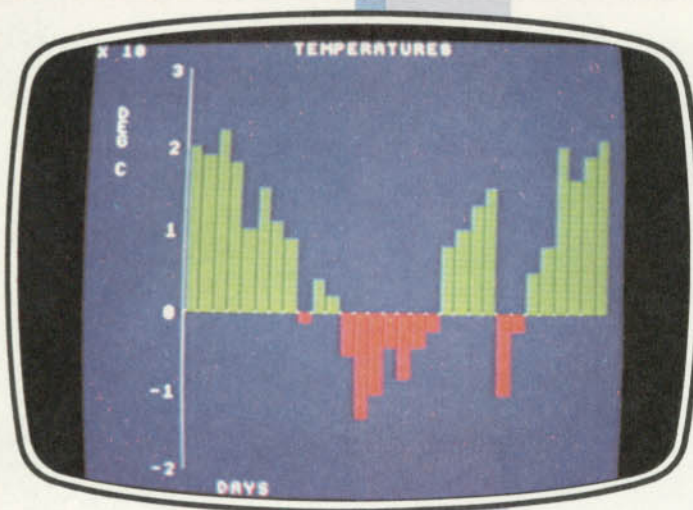
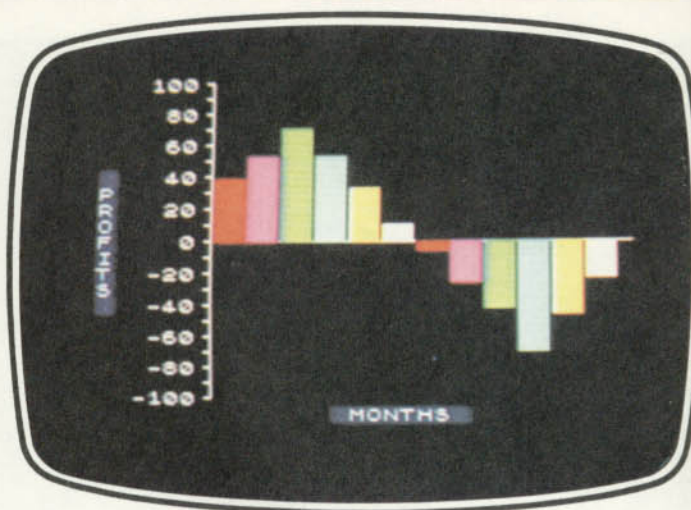
onde:

x' = valor escalado
x = valor original
xmin = valor mínimo na escala
xmax = valor máximo na escala
pix = número máximo de pixels no eixo

Em seguida, o gráfico é elaborado. Cada computador utiliza um método distinto para traçar as barras. Se você conhecer um pouco de programação BASIC, poderá, nesse ponto, alterar o programa para fazer coisas diferentes.

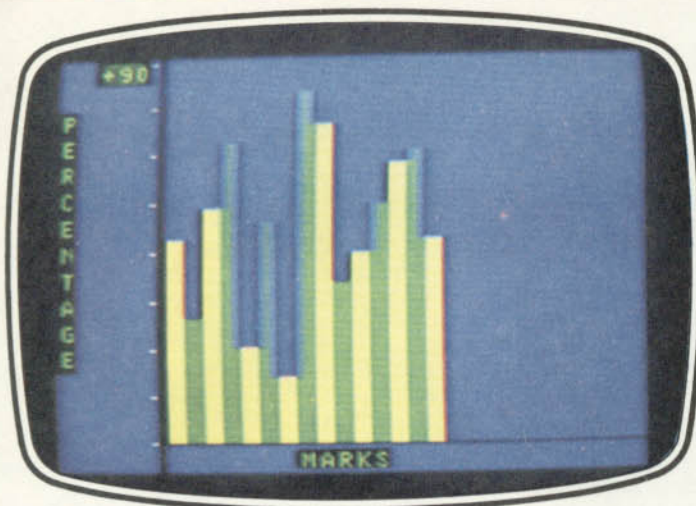
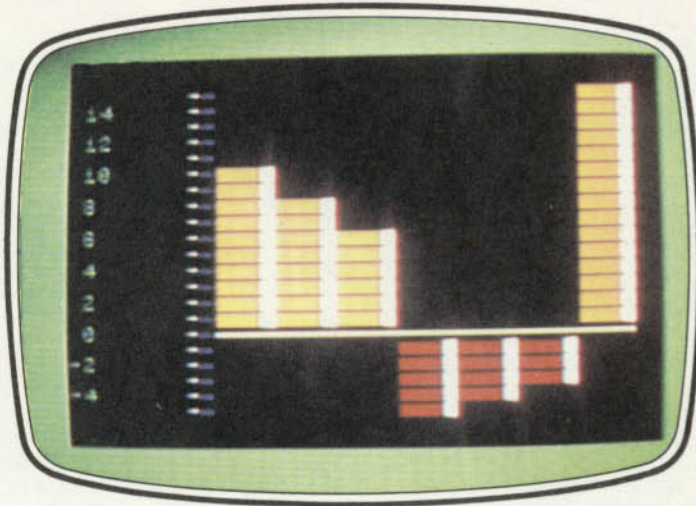
Por exemplo, a versão para o Sinclair Spectrum utiliza o comando **PLOT** para posicionar o cursor gráfico no ponto onde a barra vertical será traçada, e o comando **DRAW**, para traçá-la de uma vez. Se você substituir o comando **DRAW** por um outro **PLOT**, obterá apenas um ponto colorido, correspondente ao topo da barra.

Outra modificação se refere à definição das cores. No Sinclair Spectrum, por exemplo, o comando **INK** determina a cor a ser usada na barra. Note que a variável que se segue a este comando é modificada automaticamente pelo programa, de modo a alterar a cor da barra seguinte. Por meio de um comando **INPUT**, você pode mudar o programa, determinando que a cor seja solicitada ao usuário.

Interessante também é a possibilidade de modificar o programa para armazenar os dados entrados, em fita cassete ou disquete. Nesse caso, o menu inicial deverá ter duas opções adicionais:

MSX: gráfico da variação de temperatura ao longo do ano.

ZX Spectrum: gráfico de lucros e perdas por mês.

- armazenar dados
- gravar dados

As rotinas correspondentes a estas chamadas deverão ser programadas e adicionadas ao programa. Assim, você poderá manter séries de dados armazenados (por exemplo, os rendimentos da caderneta de poupança ou os gastos mensais com alimentação) e atualizá-los com novos valores sempre que quiser compará-los ou verificar sua tendência por meio do gráfico. Com esta modificação, você não precisará, portanto, digitar os dados anteriores toda vez que executar um programa para elaboração de gráfico.

```
10 PMODE 4,1
20 CLS
30 PRINT @45,"MENU":PRINT @102,
"1- INTRODUZIR DADOS":PRINT @16
6,"2- GRAFICO DE BARRAS":PRINT
@230,"3- VER/CORRIGIR DADOS":PR
INT @294,"4- SAIR DO PROGRAMA"
40 A$=INKEY$:IF A$<"1" OR A$>"4
" THEN 40
50 IF A$="1" AND DA=1 THEN 80
60 ON VAL(A$) GOSUB 1000,2000,3
000,4000
70 GOTO 20
80 PRINT @484,"VOCE TEM CERTEZA
? ";
90 A$=INKEY$:IF A$<>"Y" AND A$<
>"N" THEN 90
100 IF A$="Y" THEN CLEAR 200:A$
="1":GOTO 60
110 GOTO 20
1000 DA=1:CLS:INPUT"NOME DO EIX
O X";X$
1010 X$=LEFT$(X$,32):IF X$="" T
HEN X$="EIXO X"
1020 MD=INT(16-LEN(X$)/2)
1030 INPUT"NOME DO EIXO Y";Y$
1040 Y$=LEFT$(Y$,12):IF Y$="" T
HEN Y$="EIXO Y"
1050 HT=INT(6-LEN(Y$)/2)
1060 INPUT"NUMERO DE BARRAS";NB
1070 NB=INT(NB):IF NB<1 THEN 10
60
1080 BL=INT(26/NB):IF BL<1 THEN
 BL=1
1090 DIM A(NB)
1100 PRINT
1110 FOR K=1 TO NB
1120 PRINT"VALOR DA BARRA";K;:I
NPUT A(K)
1130 NEXT
1140 RETURN
2000 IF DA=0 THEN PRINT @455,"D
ADOS NAO INTRODUZIDOS":FOR K=1
TO 2000:NEXT:RETURN
2010 TP=0:BT=0
2020 FOR K=1 TO NB
2030 IF A(K)>TP THEN TP=A(K)
2040 IF A(K)<BT THEN BT=A(K)
2050 NEXT
2060 IF TP=0 AND BT=0 THEN PRIN
T "TODOS OS VALORES SAO ZERO ":
FOR K=1 TO 2000:NEXT:RETURN
2070 CLS:PRINT @64,"VOCE QUER U
M GRAFICO EM ESCALA (S/N)?"
2080 A$=INKEY$:IF A$<>"S" AND A
$<>"N" THEN 2080
2090 IF A$="N" THEN 2120
2100 IF TP>0 THEN E=INT(LOG(TP)
/LOG(10)):TP=INT(1+TP/(10^E))*1
0^E
2110 IF BT<0 THEN E=INT(LOG(-BT
)/LOG(10)):BT=-INT(1-BT/(10^E))
*10^E
2120 IN=178/(TP-BT)
2130 ST=INT(IN*TP)
2140 T$="":IF TP=0 THEN 2160
2150 IF TP>ABS(BT) THEN E=2*INT
(LOG(TP)/LOG(1000)) ELSE E=2*IN
T(LOG(ABS(BT))/LOG(1000))
2160 T$="+"+MID$(STR$(INT(.5+TP
/10^E)),2):B$="":IF BT=0 THEN 2
180
2170 B$=STR$(INT(.5+BT/10^E))
2180 SL=1:LP=NB:IF NB>26 THEN L
P=26
2190 POKE 179,243:PCLS:POKE 654
75,0:POKE 65477,0:POKE 65479,0
2200 POKE 179,2
2210 LINE(40,0)-(47,191),PRESET
,BF
2220 IF T$="" THEN 2260
2230 FOR K=1 TO LEN(T$):P=ASC(M
ID$(T$,K,1)) AND 63:FOR M=0 TO
11
2240 POKE 1540-LEN(T$)+32*M+K,P
2250 NEXT M,K
2260 IF B$="" THEN 2300
2270 FOR K=1 TO LEN(B$):P=ASC(M
ID$(B$,K,1))AND 63:FOR M=0 TO 1
1
2280 POKE 6916-LEN(B$)+M*32+K,P
2290 NEXT M,K
2300 FOR K=1 TO LEN(X$)
2310 P=ASC(MID$(X$,K,1)) AND 63
2320 FOR M=182 TO 190:POKE 1503
+MD+32*M+K,P:NEXT
2330 NEXT
2340 FOR K=1 TO LEN(Y$)
2350 FOR M=0 TO 11
2360 P=ASC(MID$(Y$,K,1)) AND 63
2370 POKE 1568+384*(K+HT)+32*M,
P
2380 NEXT M,K
2390 FOR K=5 TO 31
2400 POKE 1536+32*ST+K,128
2410 NEXT
2420 FOR K=ST TO 0 STEP -22
2430 POKE 1541+K*32,202
2440 NEXT
2450 FOR K=ST TO 178 STEP 22
2460 POKE 1541+K*32,202
2470 NEXT
2480 FOR K=SL TO LP:CO=(CO+1) A
ND 1
2490 POKE 178,35+CO*64:POKE 179
,3+CO*192
2500 IF INT(A(K)*IN)>0 THEN LIN
E(48+8*(K-SL)*BL,ST-1)-(47+8*(K
```

TRS-Color: gráfico da distribuição das notas em uma classe.

Apple: gráfico dos saldos mensais da poupança.

```
-SL+1)*BL,ST-INT(A(K)*IN)),PSET
,BF
2510 IF FIX(A(K)*IN)<0 THEN LIN
E(48+8*(K-SL)*BL,ST)-(47+8*(K-S
L+1)*BL,ST-FIX(A(K)*IN)),PRESET
,BF
2520 NEXT K
2530 IF LP=NB THEN 2570
2540 SL=LP+1:LP=LP+26:IF LP>NB
THEN LP=NB
2550 IF INKEY$="" THEN 2550
2560 POKE 179,243:LINE(48,0)-(2
55,178),PRESET,BF:GOTO 2390
2570 IF INKEY$="" THEN 2570
2580 RETURN
3000 IF DA=0 THEN PRINT @455,"D
ADOS NAO INTRODUZIDOS":FOR K=1
TO 2000:NEXT:RETURN
3010 SL=1:LP=NB:IF NB>14 THEN L
P=14
3020 CLS:PRINT"BARRA","VALOR"
3030 FOR K=SL TO LP
3040 PRINT K,A(K)
3050 NEXT
3060 IF NB=LP THEN 3130
3070 PRINT @480,"e PARA EDITAR,
 QUALQUER OUTRA   PARA CONTINUA
R";
3080 A$=INKEY$:IF A$="" THEN 30
80
3090 IF A$="E" THEN 3170
3100 SL=LP+1:LP=LP+14
3110 IF LP>NB THEN LP=NB
3120 GOTO 3020
3130 PRINT @480,"e PARA EDITAR,
 QUALQUER OUTRA   PARA RETORNAR
";
3140 A$=INKEY$:IF A$="" THEN 31
40
3150 IF A$="E" THEN 3170
3160 RETURN
3170 CLS:PRINT:INPUT"NUMERO DA
ENTRADA A SER EDITADA ?";E
3180 E=INT(E):IF E<1 OR E>NB TH
EN 3170
3190 PRINT:PRINT"NOVO VALOR DA
ENTRADA";E;:INPUT A(E)
3200 GOTO 3020
4000 CLS:PRINT @37,"VOCE TEM CE
RTEZA (S/N)?"
4010 A$=INKEY$:IF A$<>"S" AND A
$<>"N" THEN 4010
4020 IF A$="N" THEN RETURN
```

```
 10 LET g=0: POKE 23609,20:
 POKE 23658,8
 100 BORDER 7: PAPER 7: INK 0:
CLS
 110 PRINT  BRIGHT 1; PAPER 3;
INK 7;AT 4,10;" O P C O E S "
 120 PRINT  BRIGHT 1;AT 7,4;" 1
- INTRODUZIR NOVOS DADOS "
 130 PRINT  BRIGHT 1;AT 9,4;" 2
- VER / EDITAR DADOS        "
 140 PRINT  BRIGHT 1;AT 11,4;"
3- GRAFICO EM ESCALA        "
 145 PRINT  BRIGHT 1;AT 13,4;"
4- GRAFICO EM TELA CHEIA   "
 150 PRINT  BRIGHT 1; FLASH 1;
INK 2;AT 16,9;" SELECIONE OPCA
O "
 160 IF INKEY$="" THEN  GOTO
160
 170 LET A$=INKEY$: IF A$<"1"
OR A$>"4" THEN  GOTO 160
 180 GOSUB VAL A$*1000: GOTO
100
 500 REM **ROTINA PARA ENTRADA
DE DADOS NUMERICOS**
 510 INPUT (w$); LINE a$: IF
LEN a$=0 THEN  GOTO 510
 520 FOR j=1 TO LEN a$
 540 IF (a$(j)>="0" AND a$(j)<=
"9") OR a$(j)="." OR a$(j)="-"
THEN  NEXT j: LET v=VAL a$:
 RETURN
 550 GOTO 510
1000 REM **ROTINA PARA ENTRADA
DE DADOS**
1010 BORDER 1: PAPER 1: INK 7:
CLS
1020 INPUT "NOME DO EIXO X? ";
 LINE x$
1030 PRINT  INVERSE 1;AT 0,0;"
 ";x$;" "
1040 INPUT "NOME DO EIXO Y? ";
LINE y$
1050 PRINT  INVERSE 1;AT 0,16;"
 ";y$;" "
1060 LET w$="QUANTO "+x$+" (1 A
 25)? ": GOSUB 500
1070 IF v<1 OR v>25 OR v<>INT v
 THEN  GOTO 1060
1090 LET z=v: DIM a(z)
1100 FOR k=1 TO z
1110 LET w$="DIGITE DADOS PARA
"+STR$ k+" ": GOSUB 500
1120 LET a(k)=v
1130 PRINT k,a(k)
1140 NEXT k: LET g=1: PAUSE 50:
 RETURN
2000 REM **ROTINA PARA EDICAO D
E DADOS**
2010 BORDER 2: PAPER 2: INK 7
2020 LET cn=1
2025 CLS : PRINT  PAPER 6; INK
2;AT 0,0;x$,y$;TAB 31;" "
2030 PRINT cn,a(cn)
2035 PRINT #1; PAPER 6; INK 2;A
T 0,0;"EDIT para alterar valor
corrente Qualquer tecla para co
ntinuar  "
2040 PAUSE 0
2050 IF INKEY$="" THEN  GOTO 20
50
2060 LET c$=INKEY$
2070 IF c$=CHR$ 7 THEN  GOSUB 2
500
2080 IF cn=z THEN  PRINT  PAPER
 6; INK 2;"FIM DOS DADOS": PAUS
E 100: RETURN
2090 LET cn=cn+1: IF cn=21 THEN
 GOTO 2025
2100 GOTO 2030
2500 LET w$="INTRODUZIR NOVO VA
LOR PARA "+STR$ cn+" ": GOSUB 5
00
2510 LET a(cn)=v: PRINT  PAPER
6; INK 2;cn,a(cn);TAB 31;" ": R
ETURN
```

```
3000 REM **GRAFICO EM ESCALA**
3010 BORDER 0: PAPER 0: INK 7:
CLS : LET hi=0: LET lo=0
3020 FOR k=1 TO z
3030 IF a(k)>hi THEN  LET hi=a(
k)
3040 IF a(k)<lo THEN  LET lo=a(
k)
3050 NEXT k
3060 LET type=2: LET org=4
3070 IF lo<0 THEN  LET type=1:
LET org=84
3080 LET h=hi: IF ABS lo>hi THE
N  LET h=ABS lo
3090 LET ra=hi-lo
3100 IF h<=1 THEN  LET hi=1: GO
TO 3150
3110 IF h<=10 THEN  LET hi=10:
GOTO 3150
3120 IF h<=100 THEN  LET hi=100
: GOTO 3150
3130 IF h<=1000 THEN  LET hi=10
00: GOTO 3150
3140 IF h<=10000 THEN  LET hi=1
0000
3150 LET wd=INT (25/z)
3160 IF type=1 THEN  LET ft=80/
hi
3170 IF type=2 THEN  LET ft=162
/hi
3180 PLOT 56,org: DRAW 198,0
3190 PLOT 55,4: DRAW 0,160
3200 FOR n=4 TO 168 STEP 8: PLO
T 52,n: DRAW 3,0: NEXT n
3220 PRINT #1; PAPER 1;AT 0,14;
" ";x$;" "
3225 LET z$=" "+y$+" "
3230 FOR n=1 TO LEN z$
3240 PRINT AT n+(19-LEN y$)/2,0
; PAPER 1;z$(n)
3250 NEXT n
3255 LET dc=1
3260 IF type=1 THEN  GOTO 3320
3270 FOR n=hi TO 0 STEP -(hi/10
)
3275 IF n<.01 THEN  LET n=0
3280 LET n$=STR$ n
3290 PRINT AT dc,(6-LEN n$);n
3295 LET dc=dc+2
3300 NEXT n
3310 GOTO 3400
3320 FOR n=hi TO -hi STEP -(hi/
5)
3330 IF n<.01 AND n>-.01 THEN
LET n=0
3340 LET n$=STR$ n
3350 PRINT AT dc,(6-LEN n$);n
3360 LET dc=dc+2
3370 NEXT n
3400 LET ink=1
3410 FOR n=1 TO z
3420 LET cm=org
3430 LET ink=ink+1: IF ink=8 TH
EN  LET ink=2
3440 INK ink
3450 FOR m=1 TO (a(n)*ft) STEP
SGN a(n)
3460 PLOT 56+(n-1)*wd*8,cm: DRA
W wd*8-2,0
3470 LET cm=cm+SGN a(n)
3480 NEXT m
3490 NEXT n
3500 IF INKEY$<>"" THEN  GOTO 3
500
3510 IF INKEY$="" THEN  GOTO 35
10
3520 RETURN
4000 REM **GRAFICO EM TELA CHEI
A**
4010 BORDER 0: PAPER 0: INK 7:
CLS : LET hi=0: LET lo=0
4020 FOR n=1 TO z
4030 IF a(n)>hi THEN  LET hi=a(
n)
4040 IF a(n)<lo THEN  LET lo=a(
n)
4050 NEXT n
4060 LET ra=hi-lo: LET ft=175/r
a: LET org=(ra-hi)*ft
4070 LET wd=INT (25/z)
4080 PLOT 56,org: DRAW 198,0
4090 PLOT 55,0: DRAW 0,175
4100 PRINT #1; PAPER 1;AT 0,14;
" ";x$;" "
4110 LET z$=" "+y$+" "
4120 FOR n=1 TO LEN z$
4130 PRINT AT n+(19-LEN y$)/2,0
; PAPER 1;z$(n)
4140 NEXT n
4150 PRINT AT 0,0;hi;AT 21,0;lo
4200 GOTO 3400
```

```
10 CLS:RESTORE
20 DATA ENTRAR DADOS,GRÁFICO DE
 BARRAS,VER/EDITAR DADOS,FIM DE
 PROGRAMA
30 LOCATE7,2:PRINT"M E N U   P
R I N C I P A L"
40 FORI=1TO4:READM$
50 LOCATE10,2*I+7:PRINTI;"- ";M
$
60 NEXT
80 PRINT:PRINT:PRINTTAB(20)"OPÇ
ÃO?";
90 R$=INKEY$:IFR$=""THEN90
100 IFR$<"1"ORR$>"4"THEN90
110 ONVAL(R$)GOTO1000,2000,3000
,4000
150 LO=VA(1):HI=VA(1)
160 FORI=1TONB%
170 IFVA(I)>HITHENHI=VA(I)
180 IFVA(I)<LOTHENLO=VA(I)
190 NEXT:RETURN
1000 CLS:IFNB%<>0THENLOCATE10,1
4:PRINT"CONFIRMA? ";ELSE1030
1010 R$=INKEY$:IFR$=""THEN1010
1020 IFR$<>"S"THEN10
1030 CLS:LOCATE10:PRINT"ENTRADA
 DE DADOS"
1040 CLEAR:LOCATE0,4:INPUT"NOME
 DO EIXO-X ";X$
1050 X$=LEFT$(X$,9):IFX$=""THEN
X$="EIXO-X"
1060 INPUT"NOME DO EIXO-Y ";Y$
1070 Y$=LEFT$(Y$,9):IFY$=""THEN
Y$="EIXO-Y"
1080 PRINT:INPUT"NÚMERO DE BARR
AS: ";NB%
1090 IFNB%>65THEN1080
1100 DIMVA(NB%):PRINT:PRINT
1110 FORI=1TONB%
1120 PRINT"VALOR DA BARRA ";I;:
INPUTVA(I)
1130 NEXT:GOTO10
2000 IFNB%=0THEN10
2010 GOSUB150
2020 IFSGN(HI)<>SGN(LO)THENHT=A
BS(HI)+ABS(LO):GOTO2050
2030 HT=ABS((SGN(HI)=1)*ABS(HI)
+(SGN(LO)=-1)*ABS(LO))
2040 IFHT=0THEN10
2050 A=2*INT((180-3*NB%)/NB%/2)
:S=3:ES=150/HT
2060 IFA>15THENA=15
2070 IFLO>=0THENY=155:GOTO2100
2080 IFHI<=0THENY=5:GOTO2100
2090 Y=INT(80+(ABS(HI)-ABS(LO))
*ES/2)
2100 SCREEN2:COLOR 15,4,4:P=61
2110 LINE (55,5)-(55,158):LINE
(52,Y)-(250,Y)
2120 FORI=1TONB%
2130 LINE (P,Y-SGN(VA(I)))-(P+A
,INT(Y-(VA(I)*ES))),6,BF
2140 P=P+S+A:NEXT
2150 OPEN "GRP: " FOR OUTPUTAS#
1
2160 PRESET(0,3)
2170 FORI=1TOLEN(Y$)
2180 PRINT#1,TAB(1)MID$(Y$,I,1)
:NEXT
2190 PRESET (170,160)
2200 PRINT#1,X$
2210 IFSGN(LO)<>SGN(HI)THEN2250
2220 PRESET(0,Y):PRINT#1,TAB(4)
"0"
2230 PRESET(15,155+(150*(HI>0))
):PRINT#1,LO*(-(SGN(LO)=-1))+HI
*(-(SGN(HI)=1))
2240 GOTO 2270
2250 PRESET(15,155):PRINT#1,LO:
PRESET(15,5):PRINT#1,HI
```

```
2260 PRESET(0,Y):PRINT#1,TAB(4)
"0"
2270 CLOSE
2280 IFINKEY$=""THEN2280
2290 SCREEN0:GOTO10
3000 IFNB%=0THEN10
3010 CLS:LOCATE10:PRINT"MÓDULO
DE EDIÇÃO"
3020 PRINT:PRINT"BARRA","VALOR"
3030 FORJ=1TONB%
3040 PRINTJ,VA(J)
3050 IFINT(J/17)=J/17THENGOSUB3
090
3060 NEXT
3070 GOSUB3090
3080 GOTO10
3090 LOCATE0,20:PRINT"Tecle <E>
 para editar ou qualquer outra
   tecla para continuar";
3100 R$=INKEY$:IFR$=""THEN3100
3110 IFR$<>"E"THENLOCATE0,3:FOR
H=1TO19:PRINTSPC(40):NEXT:LOCAT
E0,3:RETURN
3120 LOCATE0,20:PRINTSPC(79):LO
CATE0,20:INPUT"Qual barra ";B
3130 LOCATE0,21:INPUT"Novo valo
r ";VA(B)
3140 I=B:GOSUB150
3150 GOTO3090
4000 CLS:LOCATE7,14:INPUT"FIM D
E PROGRAMA (S/N)";R$
4010 IFR$<>"S"THEN10
```

```
10  HOME : RESTORE
20  DATA  ENTRAR DADOS,GRAFICO
DE BARRAS,VER / EDITAR DADOS, F
IM DE PROGRAMA
30  VTAB 3: HTAB 8: PRINT "M E
N U    P R I N CI P A L"
40  FOR I = 1 TO 4
50  READ M$
60  HTAB 10: VTAB 2 * I + 8: PR
INT I;"- ";M$
70  NEXT
80  PRINT : PRINT : HTAB 20: PR
INT "OPCAO =>";
90  GET R$: IF R$ < "1" OR R$ >
 "4" THEN 90
100  ON  VAL (R$) GOTO 1000,200
0,3000,4000
110 LO = VA(1):HI = VA(1)
120  FOR I = 1 TO NB%
130  IF VA(I) > HI THEN HI = VA
(I)
140  IF VA(I) < LO THEN LO = VA
(I)
150  NEXT : RETURN
1000  HOME : IF NB% <  > 0 THEN
  VTAB 15: PRINT  TAB( 10)"CONF
IRMA? (S/N)";: GET R$: IF R$ <
 > "S" THEN 10
1010  HOME : PRINT  TAB( 10)"EN
TRADA DE DADOS"
1020  CLEAR : VTAB 4: INPUT "NO
ME DO EIXO 'X': ";X$
1030 X$ =  LEFT$ (X$,5): IF X$
= "" THEN X$ = "EIXO-X"
1040  INPUT "NOME DO EIXO 'Y':
";Y$
1050 Y$ =  LEFT$ (Y$,5): IF Y$
= "" THEN Y$ = "EIXO-Y"
1060  PRINT : INPUT "NUMERO DE
BARRAS: ";NB%
1070  IF NB% > 65 THEN 1060
1080  DIM VA(NB%)
1090  PRINT : PRINT
1100  FOR I = 1 TO NB%
1110  PRINT "VALOR DA BARRA ";I
;: INPUT VA(I)
1120  NEXT : GOTO 10
2000  IF NB% = 0 THEN 10
2010  GOSUB 110
2020  IF  SGN (HI) <  >  SGN (L
O) THEN HT =  ABS (HI) +  ABS (
LO): GOTO 2050
2030 HT =  ABS (( SGN (HI) - 1)
 *  ABS (HI) + ( SGN (LO) =  -
1) *  ABS (LO))
2040  IF HT = 0 THEN 10
2050 A = 2 *  INT ((270 - 2 * N
B%) / NB% / 2):S = 2:ES = 150 /
 HT
2060  IF A > 15 THEN A = 15
2070  IF LO >  = 0 THEN Y = 155
: GOTO 2100
2080  IF HI <  = 0 THEN Y = 5:
GOTO 2100
2090 Y =  INT (76 + ( ABS (HI)
-  ABS (LO)) * ES / 2)
2100  HGR : HCOLOR= 7:P = 4
2110  HPLOT 2,0 TO 2,158: HPLOT
 0,Y TO 279,Y
2120  HCOLOR= 5
2130  FOR I = 1 TO NB%
2140  FOR J = 1 TO A
2150  HPLOT J + P,Y -  SGN (VA(
I)) TO J + P, INT (Y - (VA(I) *
 ES)) + 1
2160  NEXT :P = P + S + A: NEXT

2170  VTAB 21: HTAB 1: PRINT X$
; TAB( 34);Y$
2180  VTAB 22: HTAB 10: PRINT "
VAL MAX=";HI;: HTAB 25: PRINT "
VAL MIN=";LO
2190  PRINT  TAB( 5);"PRESSIONE
 QUALQUER TECLA PARA SAIR";
2200  GET R$: TEXT : GOTO 10
3000  IF NB% = 0 THEN  GOTO 10
3010  HOME : PRINT  TAB( 10)"MO
DULO DE EDICAO"
3020  PRINT : PRINT "BARRA","VA
LOR"
3030  FOR J = 1 TO NB%
3040  PRINT J,VA(J)
3050  IF  PEEK (37) = 20 THEN
GOSUB 3090
3060  NEXT
3070  GOSUB 3090
3080  GOTO 10
3090  VTAB 21: HTAB 1: PRINT "T
ECLE <E> PARA EDITAR OU QUALQUE
R OUTRA    TECLA PARA CONTINUAR
";: GET R$: IF R$ <  > "E" THEN
  VTAB 4: HTAB 1: CALL  - 958:
RETURN
3100  VTAB 21: HTAB 1: CALL  -
958: INPUT "QUAL BARRA? ";B
3110  INPUT "NOVO VALOR? ";VA(B
)
3120 I = B: GOSUB 110
3130  GOTO 3090
4000  HOME : VTAB 15: HTAB 10:
PRINT "FIM DE PROGRAMA? ";: GET
 R$: IF R$ <  > "S" THEN 10
```

CRIE SPRITES NO MSX

Entre os vários recursos gráficos do MSX, o sprite destaca-se por servir de base ao funcionamento da maioria dos jogos de ação. Veja como é fácil usá-lo.

Um sprite é um tipo especial de caractere gráfico definido pelo usuário, desenhado em alta resolução e extremamente fácil de movimentar. Alguns exemplos de seu uso já foram vistos em artigos anteriores. Além da mobilidade, o sprite tem outras características especiais, tais como a possibilidade de definir e movimentar mais de um sprite e de indicar se houve colisão na tela — ou seja, se dois sprites coincidiram no mesmo ponto.

Não surpreende, portanto, que os sprites estejam presentes na maioria dos programas de jogos. Mas são, também, utilizados em qualquer tipo de programa que necessite de figuras móveis em alta resolução — por exemplo, programas financeiros que empreguem um "ícone" (símbolo apontando para as opções).

DEFINIÇÃO DE UM SPRITE

O desenho de um sprite é definido informando-se ao computador quais pontos devem ser "acesos" e quais devem ser "apagados", para que se obtenha o padrão desejado. Existem dois tamanhos de sprite: pequeno (8x8 pontos) e grande (16x16). No decorrer deste artigo, trataremos apenas dos sprites grandes, salvo menção em contrário.

Um sprite grande é formado por dezesseis linhas de dezesseis pontos cada. Em vez de se definir, ponto por ponto, quais serão acesos ou apagados, reúnem-se cada oito pontos em um grupo. Assim, a informação sobre o padrão de uma linha do sprite é armazenada em dois grupos de oito pontos cada. Atribuindo-se o algarismo "1" para os pontos acesos e "0" para os apagados, cada grupo de oito pontos se transformará em um número binário (byte de 8 bits) que, convertido para a forma decimal, dará origem a um número entre 0 e 255. Cada sprite, portanto, é formado por 32 grupos de oito pontos, 32 números binários de oito dígitos (8 bits ou 1 byte) ou, simplesmente, 32 números. Estes números são agrupados em uma linha **DATA**, ordenados como descrito adiante. Calculam-se da mesma maneira os valores para sprites pequenos, já que eles são formados por oito linhas de oito pontos cada, dando origem a 8 bytes.

A ordem dos bytes no sprite é a seguinte:

Linha 1: BYTE 1 BYTE 17
Linha 2: BYTE 2 BYTE 18
Linha 3: BYTE 3 BYTE 19

... e assim por diante até que:

Linha 15: BYTE 15 BYTE 31
Linha 16: BYTE 16 BYTE 32

... sendo que os bytes foram numerados na ordem em que aparecerão na linha **DATA**.

Uma orientação para o cálculo destes valores, uma vez definido o padrão gráfico para o sprite, é dada na figura da página 190. Por meio do exemplo, você terá uma idéia de como obter o valor decimal correspondente a um grupo de oito pontos. Se todos os pontos estiverem acesos, o valor decimal será

128 + 64 + 32 + 16 + 8 + 4 + 2 + 1, ou seja, 255 (veja, no alto da figura, o valor pelo qual se deve multiplicar cada ponto). Se todos estiverem apagados, a soma dos valores será zero. Entre estes dois extremos haverá um único valor para cada permutação possível de pontos acesos e apagados. Cada valor obtido pode ser usado na definição do sprite, desde que colocado em uma linha **DATA**, na ordem já especificada.

OS PRIMEIROS PASSOS

Exercite-se um pouco. Comece desenhando o padrão da figura desejada em papel quadriculado. Delimite uma área de dezesseis por dezesseis quadrinhos e indique as posições dos bytes. Na figura da página 190, por exemplo, definimos um sprite que representa uma nave espacial. Ao lado da figura, temos o valor de cada byte em decimal. O primeiro byte da primeira linha tem todos os seus pontos apagados, exceto o último, cujo valor podemos encontrar no alto da figura. Como só há um ponto aceso, seu valor é o valor total do byte: 1. Na segunda linha temos o segundo byte (veja a ordem dos bytes no sprite, já fornecida). O segundo ponto está aceso, bem como o último. Se consultarmos os valores desses pontos no alta da tabela, veremos que o valor do byte 2 é 64 + 1 = 65. Os próximos três bytes têm o mesmo valor. A propósito, uma recomendação prática: após calcular o valor de um byte, verifique se, no resto do sprite, existem outros bytes com o mesmo padrão e, portanto, com o mesmo valor. Agora vamos à linha 6. O byte 6 tem o segundo e o sétimo pontos acesos, o que lhe dá um valor de 64 + 2 = 66. O valor do byte 7, que apresenta os pontos 2 e 6 acesos, é 64 + 4 = 68. O byte 8, por sua vez, tem os pontos 1, 2, 3, 5 e 8 acesos, resultando num valor de 128 + 64 + 32 + 8 + 1 = 233. Calcule os bytes restantes do mesmo modo. Terminado o trabalho, coloque os números obtidos em uma linha com instrução **DATA**. Para definir e movimentar o sprite, digite o programa completo:

```
10 SCREEN 2,2:COLOR 15,4,4
20 FOR I=1 TO 32
30 READ A:A$=A$+CHR$(A)
40 NEXT I
50 SPRITE$(0)=A$
60 FOR I=210 TO -20 STEP -.5
70 PUT SPRITE 1,(I,I),10,0
80 NEXT I
100 DATA 1,65,65,65,65,66,68,
233,169,191,175,163,227,3,15,17
,0,4,4,4,4,132,68,46,42,250,234
,138,142,128,224,16
```

Para a definição de um sprite em BASIC, devem-se colocar os valores já calculados dentro de uma cadeia alfanumérica. Em nosso exemplo, isso é feito pelo laço **FOR...NEXT** das linhas 20, 30 e 40. Note que dentro de uma cadeia não se colocam números e, sim, caracteres cujos códigos ASCII correspondam aos números desejados. Observe a linha 30. Ela lê os dados da linha **DATA** e os coloca na variável **A$**, usando a função **CHRS (A)**, que define o caractere cujo código está na variável **A**.

A definição propriamente dita do sprite é feita na linha 50, pelo comando **SPRITES (0) A$**. O número 0 é o número do sprite (no MSX podemos ter sprites numerados de 0 a 9). Observe que na linha 10 escolhemos a tela de alta resolução, pois a de texto de 40 colunas não permite o uso de sprites. Nessa linha encontramos o comando **SCREEN 2,2.** Como você já deve saber, o primeiro número corresponde ao formato da tela (2 para a tela de alta resolução). O segundo número deve ser novo para você. Ele estabelece o tamanho do sprite utilizado: 0-pequeno (8x8), 1-pequeno ampliado (16x16, baixa resolução), 2-grande (16x16) e 3-grande ampliado (32x32, baixa resolução).

Para desenhar e movimentar o sprite na tela usamos o comando **PUT SPRITE.** Não é preciso seguir o sprite apagando posições já ocupadas: ao desenharmos um sprite na nova posição, a antiga é automaticamente apagada. O formato desse comando é **PUT SPRITE P, (X, Y), C, N,** onde **P** é a *prioridade* do sprite, **X** é a *coordenada horizontal* do sprite, variando de -32 (fora da tela, portanto) a 255; **Y** é a *coordenada vertical*, que varia de -32 a 191, podendo ainda assumir os valores 208, que faz com que todos os sprites de menor prioridade desapareçam da tela, e 209, que apaga o sprite na tela; **C** é a *cor* do sprite, e **N**, o *número* do sprite.

Quando dois ou mais sprites ocuparem a mesma posição na tela, aparecerá apenas o que estiver "na frente", ou seja, aquele que tiver o número de prioridade menor. Isso permite dar aos seus jogos efeitos tridimensionais. Os carros passam atrás dos postes, e ursos se escondem atrás de árvores, por exemplo.

```
         VALORES DOS PONTOS           NUMEROS DA
                                      LINHA DATA
           128 64 32 16 8 4 2 1 128 64 32 16 8 4 2 1
LINHAS   1 .......X........              1     0
         2 .X......X......X..            65    4
         3 .X......X......X..            65    4
         4 .X......X......X..            65    4
         5 .X......X......X..            65    4
         6 .X.....X.X.....X..            66  132
         7 .X...X...X...X..              68   68
         8 XXX.X..X..X.XXX.              233   46
         9 X.X.X..X..X.X.X.              169   42
        10 X.XXXXXXXXXXX.X.              191  250
        11 X.X.XXXXXXX.X.X.              175  234
        12 X.X...XXX...X.X.              163  138
        13 XXX...XXX...XXX.              227  142
        14 ......XXX.......                3  128
        15 ....XXXXXXX.....               15  224
        16 ...X...X...X....               17   16
```

Na figura ao lado — um sprite representando uma nave espacial —, os números das linhas e os valores dos pontos estão indicados em suas respectivas posições, para facilitar o acompanhamento dos cálculos. Veja como os valores são somados, para dar origem aos números da linha *DATA*, mostrados à direita. Há dois números para cada linha, representando dois bytes com oito pontos cada. O cálculo desses valores pode ser feito manualmente ou com o uso de um programa.

UM PROGRAMA PARA CRIAR SPRITES

A tarefa mais aborrecida, ao se trabalhar com sprites, consiste em calcular os 32 números que serão colocados na linha **DATA**. Em vez de fazê-lo manualmente, use um programa que trabalhe por você.

Este deverá, em primeiro lugar, permitir a utilização dos recursos de edição e movimentação do cursor do MSX para desenhar a figura do sprite na tela. Em seguida, precisará calcular os valores correspondentes que serão incluídos na linha da instrução **DATA**, possibilitando-lhe usar o sprite em seus programas.

O programa seguinte foi preparado para efetuar todas essas coisas, oferecendo ainda a opção de produzir uma cópia impressa, caso você tenha uma impressora. A tarefa de digitá-lo e gravá-lo em uma fita cassete é compensadora, pois facilita muito a posterior criação de sprites.

```
5 SCREEN 0:KEY OFF
10 DIM A$(16),Z(2,16),A(16)
20 CLS:PRINT TAB(10);"EDITOR DE
 SPRITES"
30 PRINT:PRINT TAB(8);:INPUT "C
ÓPIA  IMPRESSA (S/N)";I$:IF I$=
"S" THEN PR$="S"
40 IF I$<>"S" AND I$<>"N" THEN
GOTO 20
50 PRINT:PRINT TAB(15);"Aguarde
"
60 FOR Z=1 TO 8
62 READ A(Z):A(Z+8)=A(Z)
64 NEXT
66 DATA 128,64,32,16,8,4,2,1
70 FOR Z=1 TO 16
75 READ A$(Z)
80 FOR ZZ=1 TO 8
85 IF MID$(A$(Z),ZZ,1)="*" THEN
 Z(1,Z)=Z(1,Z)+A(ZZ)
90 NEXT ZZ:FOR ZZ=9 TO 16
95 IF MID$(A$(Z),ZZ,1)="*" THEN
 Z(2,Z)=Z(2,Z)+A(ZZ)
100 NEXT ZZ,Z
110 CLS:IF PR$="S" THEN OPEN "L
PT:" FOR OUTPUT AS #1 ELSE OPEN
 "CRT:" FOR OUTPUT AS #1
120 PRINT #1,"Linha DATA ";
130 FOR Z=1 TO 16
132 PRINT #1,Z(1,Z);",";
134 NEXT Z
136 FOR Z=1 TO 16
138 PRINT #1,Z(2,Z);:IF Z<16 TH
EN PRINT #1,",";
140 NEXT Z
145 PRINT #1," ":PRINT #1," "
150 IF PR$="S" THEN GOTO 170
155 PRINT #1,"Aperte RETURN par
a continuar"
160 K$=INKEY$
165 IF K$<>CHR$(13) THEN GOTO 1
60
170 SCREEN 1,3:COLOR 14,5,5
172 VPOKE BASE(6)+15,68
174 VPOKE BASE(6)+5,69
176 PRINT #1,TAB(5);"DESENHO DO
 SPRITE"
178 PRINT #1," ":PRINT #1," "
180 PRINT #1,TAB(5);"1234567890
123456"
190 FOR Z=1 TO 16
195 PRINT #1,TAB(5);
200 FOR ZZ=1 TO 16
205 IF MID$(A$(Z),ZZ,1)="*" THE
N PRINT #1,CHR$(120); ELSE PRIN
T #1,".";
210 NEXT ZZ:PRINT #1," ";Z
220 NEXT Z:IF PR$="S" THEN CLOS
E:END
230 FOR Z=1 TO 16
240 A$=A$+CHR$(Z(1,Z))
250 B$=B$+CHR$(Z(2,Z))
260 NEXT:SPRITE$(0)=A$+B$
270 FOR Z=190 TO 5 STEP -5
275 PUT SPRITE0,(10,Z),1
280 NEXT
290 IF PR$<>"S" THEN GOTO 290
300 REM  1234567890123456
301 DATA"                "
302 DATA"                "
303 DATA"                "
304 DATA"                "
305 DATA"                "
306 DATA"                "
307 DATA"                "
308 DATA"                "
309 DATA"                "
310 DATA"                "
311 DATA"                "
312 DATA"                "
313 DATA"                "
314 DATA"                "
315 DATA"                "
316 DATA"                "
```

Rode o programa, tal como ele está, para verificar se houve algum erro de digitação. Ao fazê-lo, responda N à pergunta da linha 30.

Para usar o programa, digite **LIST 300-**. Isso fará com que as dezesseis linhas **DATA** vazias do final da listagem surjam na tela. Desenhe dentro delas o padrão do seu sprite. Cada uma das dezesseis posições das linhas, entre as aspas, representa um ponto. Use apenas os controles de movimentação do cursor e a barra de espaço, colocando um asterisco nas posições que deseja acender. Evite usar **INS** e **DEL**, que alteram o número de posições entre as aspas. Não deixe de apertar **RETURN** em cada uma das dezesseis linhas **DATA**; caso contrário, o MSX não se lembrará delas. Oriente-se pelos números da linha 300, que correspondem às posições horizontais. Um desenho típico se encontra na figura da página ao lado, que representa um pássaro voando.

Quando completar seu desenho, apague a tela e liste as últimas linhas para conferir. Rode o programa. O computador lhe dará, então, a opção de cópia impressa. Responda N e veja o resultado na tela.

Primeiro, surgem os 32 números das linhas **DATA**. Um desenho ampliado de

PLANEJAMENTO DE SPRITES

Existe um modo muito prático de usar o programa de criação de sprites. Desenhe seu sprite em papel quadriculado com tamanho correspondente ao das linhas **DATA** do final da listagem. Pregue-o com fita adesiva na tela de sua TV. Em seguida, simplesmente use os controles do cursor para desenhar asteriscos nas posições dos quadrinhos que você pintou.

seu sprite é mostrado a seguir, apertando-se a tecla **RETURN**. As teclas **CTRL** e **STOP**, pressionadas ao mesmo tempo, interrompem o programa. Caso tenha uma impressora e deseje uma cópia impressa, rode o programa novamente e responda S à pergunta da linha 30. O resultado será enviado à impressora, e não à tela. Algumas impressoras requerem um ajuste no número máximo de caracteres por linha (use quarenta).

Para o exemplo da figura do pássaro, o resultado é o seguinte:

```
Linha DATA  128 , 192 , 176 , 7
2 , 116 , 84 , 50 , 42 , 41 , 3
7 , 26 , 4 , 56 , 199 , 98 , 28
 , 7 , 30 , 42 , 84 , 84 , 168
, 168 , 208 , 160 , 248 , 4 , 3
4 , 222 , 33 , 144 , 72

      DESENHO DO SPRITE


      1234567890123456
      x............xxx   1
      xx.........xxxx.   2
      x.xx......x.x.x.   3
      .x..x....x.x.x..   4
      .xxx.x...x.x.x..   5
      .x.x.x..x.x.x...   6
      ..xx..x.x.x.x...   7
      ..x.x.x.xx.x....   8
      ..x.x..xx.x.....   9
      ..x..x.xxxxxx...  10
      ...xx.x......x..  11
      .....x....x...x.  12
      ..xxx....xx.xxxx. 13
      xx...xxx..x....x  14
      .xx...x.x..x....  15
      ...xxx...x..x...  16
```

O mesmo programa pode ser utilizado para sprites pequenos. Para tanto, desenhe seu padrão utilizando apenas os oito primeiros bytes, ou seja, no quadrante superior esquerdo do conjunto de linhas **DATA**. Considere apenas os primeiros oito números calculados.

COMO USAR O SPRITE

O programa seguinte desenha e movimenta o sprite do pássaro, que acabamos de calcular.

```
10 SCREEN 1,2:COLOR 15,4,4
20 FOR I=1 TO 32
21 READ A:A$=A$+CHR$(A)
22 NEXT I:TIME=0
23 SPRITE$(0)=A$
25 X=90:Y=X:GOTO 50
30 K$=INKEY$:A=0:XX=0
31 IF K$="" THEN GOTO 30
32 IF K$=CHR$(30) THEN A=1:GOTO
 50
35 IF K$=CHR$(31) THEN A=2:GOTO
 50
40 IF K$=CHR$(29) THEN XX=-2:GO
TO 50
45 IF K$=CHR$(28) THEN XX=2:GOT
O 50
50 FOR Z=1 TO 10
52 X=X+XX:IF X>255 THEN X=0
55 IF X<-32 THEN X=255
60 IF A=1 AND Y>-32 THEN Y=Y-2
65 IF A=2 AND Y<190 THEN Y=Y+2
70 PUT SPRITE 0,(X,Y),15
75 NEXT Z:GOTO 30
100 DATA 128,192,176,72,116,84,
50,42,41,37,26,4,56,199,98,28,7
,30,42,84,84,168,168,208,160,24
8,4,34,222,33,144,72
```

O sprite é desenhado na tela de textos de 32 colunas e pode ser movimentado pelas teclas de controle do cursor. Para usar qualquer outro sprite, basta mudar a linha 100.

O laço contido nas linhas 20, 21 e 22 lê os valores da linha 100 e os coloca na variável **A$**. A linha 25 define o sprite atribuindo-lhe o número 0. As linhas de 30 a 75 movimentam o sprite conforme a tecla pressionada.

COLISÃO DE SPRITES

Outro recurso interessante dos sprites é a indicação da ocorrência de colisão entre dois deles, em sua movimentação pela tela. Para entender como isso é feito, apague a linha 32 do último programa e acrescente:

```
20 SPRITE ON:FOR I=1 TO 40
22 NEXT I:TIME=0:S=0
23 SPRITE$(0)=LEFT$(A$,32)
24 SPRITE$(1)=RIGHT$(A$,8)
25 X=90:Y=X:XX=100:YY=XX
30 FOR Z=1 TO 5:LOCATE 0,0
31 PRINT "TEMPO: ";TIME;
35 A=INT(RND(1)*3)+1:X=X+10
36 IF X>255 THEN X=-30
40 IF A=1 AND Y>-30 THEN Y=Y-10
45 IF A=2 AND Y<250 THEN Y=Y+10
50 PUT SPRITE 1,(X,Y),15,0
52 Z$=INKEY$:IF Z$="" THEN GOTO
 52
53 IF Z$=CHR$(29) AND XX>-30 TH
EN XX=XX-5
55 IF Z$=CHR$(28) AND XX<250 TH
EN XX=XX+5
60 IF Z$=CHR$(30) AND YY>-30 TH
EN YY=YY-5
65 IF Z$=CHR$(31) AND YY<190 TH
EN YY=YY+5
70 PUT SPRITE 0,(XX,YY),10,1
72 ON SPRITE GOSUB 1000
75 LOCATE 20,0:PRINT "SCORE: ";
S
76 NEXT Z:IF TIME<5000 THEN 30
80 PRINT:PRINT "O Tempo acabou
!":END
110 DATA 8,8,62,8,8,0,0,0
1000 SPRITE OFF:S=S+1
1010 FOR I=Y TO 191
1020 PUT SPRITE 1,(X,I),15,0
1030 NEXT I:SPRITE ON
1040 X=-30:RETURN
```

COMO EDITAR UMA LINHA DATA

Para evitar o trabalho de digitar a linha **DATA** criada pelo programa, faça o seguinte: assim que o computador fornecer os números, pare o programa usando **CTRL** + **STOP**. Movimente o cursor até o topo da tela, onde se encontra a linha **DATA**. Usando **DEL**, apague a palavra "Linha" e, usando **INS**, escreva o número da linha **DATA** no programa que está preparando. A seguir, aperte **RETURN**. O efeito será idêntico ao da digitação de todos aqueles números.

Este método pode ser usado para colocar a mesma linha em programas já existentes em fita. Carregue o programa sem apagar a linha **DATA** da tabela. O programa carregado apaga a memória, mas não apaga a tela. Movimente o cursor até a linha **DATA**, modifique seu número, se necessário, e aperte **RETURN**.

Este é um pequeno jogo de "tiro ao pássaro", extremamente simples, mas útil à nossa finalidade. Não explicaremos seu funcionamento enquanto jogo, restringindo-nos apenas ao controle de colisões.

Para que possamos trabalhar com colisões de sprites, devemos "habilitar" o programa a reconhecer tal ocorrência, por meio da instrução **SPRITE ON**, na linha 20. Uma vez feita a habilitação, devemos usar a instrução **ON SPRITE GOSUB**, que desvia o programa para uma sub-rotina sempre que quisermos verificar se houve uma colisão de sprites. Nesta sub-rotina pode estar programado um efeito visual ou sonoro de explosão, por exemplo. Em nosso programa, tal comando ocorre na linha 72. Caso haja colisão entre o pássaro e a "mira", o controle é transferido para a sub-rotina 1000, que incrementa o escore e faz com que o pássaro caia. Durante a execução da sub-rotina, o programa é desabilitado pela instrução **SPRITE OFF** na linha 1000. Esse é um procedimento usual e quase sempre necessário, e não há uma explicação simples para ele. Retire este comando, bem como o de habilitação da linha 1030, e veja como ocorrem erros no escore.

Existe ainda a instrução **SPRITE STOP**, que não desabilita o programa mas adia qualquer desvio, guardando a ocorrência de colisões na memória. Só haverá o desvio após a ocorrência de um **SPRITE ON** no programa.

CONJUNTOS: CAIXAS DE INFORMAÇÃO

Os conjuntos constituem um recurso muito eficaz para armazenar e processar listas de qualquer natureza: nomes, datas, as mais diversas estatísticas. Aprenda a utilizá-los.

Os conjuntos são uma forma de representação de dados em programa BASIC, constituindo método mais adequado para se programar a manipulação de informações inter-relacionadas — por exemplo, listas dos membros de um clube, com seus respectivos endereços e inscrições, registros financeiros ou, então, listas de personagens, armas e tesouros para um jogo de aventuras.

O comando **LET**, que você já conhece, poderia ser utilizado para atribuir valores como esses a uma série de variáveis. No entanto, para listas muito longas, ele é bastante ineficiente, pois toma muito tempo de digitação.

O conjunto simplifica o trabalho, armazenando a informação de uma forma mais compacta. Em vez de uma variável diferente para cada item da informação, utiliza-se uma variável com o mesmo nome para todos os itens — o **A**, por exemplo. E, para se diferenciar um item dos demais, acrescenta-se simplesmente um número (ou, às vezes, uma expressão aritmética) entre parênteses, logo após o nome da variável. O primeiro elemento é chamado de **A(1)**, o segundo de **A(2)**, o terceiro de **A(3)** e assim por diante.

O conjunto não apenas torna o sistema de armazenagem mais compacto, como também facilita grandemente a programação da entrada, modificação e listagem dos dados nele contidos.

COMO DIMENSIONAR UM CONJUNTO

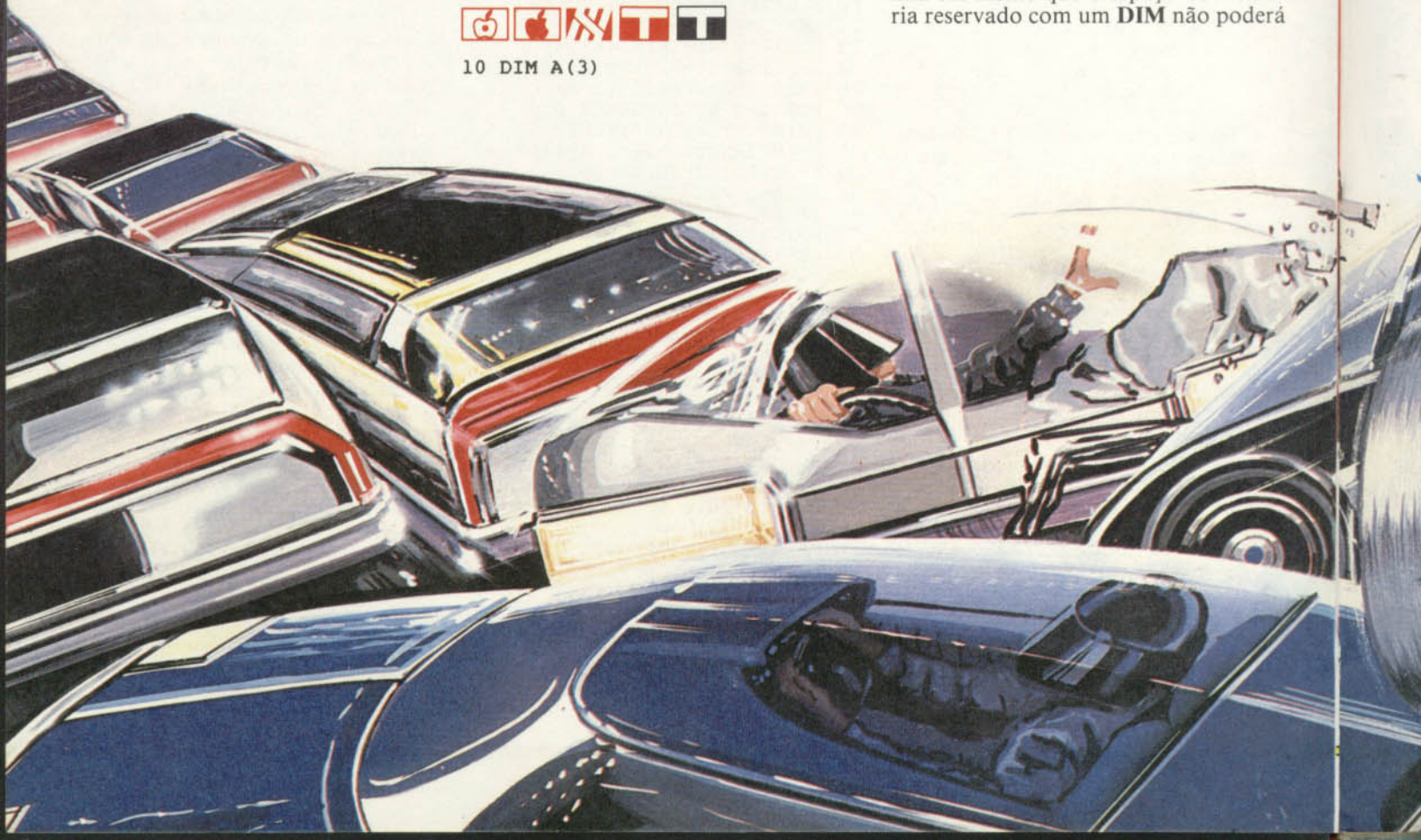
Antes de utilizar um conjunto em BASIC, é necessário declarar ao computador o tamanho (número de elementos) que ele o terá. Assim, o computador poderá reservar espaço suficiente na memória. Isto é feito mediante a declaração **DIM** (de "dimensão"), como se vê a seguir:

```
10 DIM A(3)
```

```
10 DIM a(4)
```

Use um A maiúsculo para os micros da linha ZX-81.

A declaração **DIM** acima informa ao computador que o conjunto **A** terá quatro elementos, ou variáveis. Nos computadores Apple, TRS-Color e MSX eles são numerados de **A(0)** a **A(3)**. Nos micros da linha Sinclair, que não aceitam o elemento **A(0)**, eles são numerados de **A(1)** a **A(4)**.

O número de elementos que se vai dimensionar pode ser tão grande quanto se queira (milhares, por exemplo), desde que caiba na memória RAM do computador, é claro. Na verdade, não é obrigatório dimensionar o número exato de elementos que se utilizarão: para ter a garantia de que o espaço será suficiente, reserve um pouco mais. Mas tenha em mente que o espaço de memória reservado com um **DIM** não poderá

ser ocupado por outras variáveis. Por isso, não dimensione exageradamente, ou receberá uma mensagem de "estouro de memória" (*out of memory*).

ATRIBUIÇÃO DE VALORES

O passo seguinte consiste em atribuir os valores para cada elemento. Se, por exemplo, as variáveis representassem as locações da tela nas quais se imprimiria um texto ou um gráfico, a próxima linha seria mais ou menos isto:

```
20 LET A(0)=0:LET A(1)=2:LET A(
2)=10:LET A(3)=20
```

```
20 LET a(1)=0: LET a(2)=2:
LET a(3)=10: LET a(4)=20
```

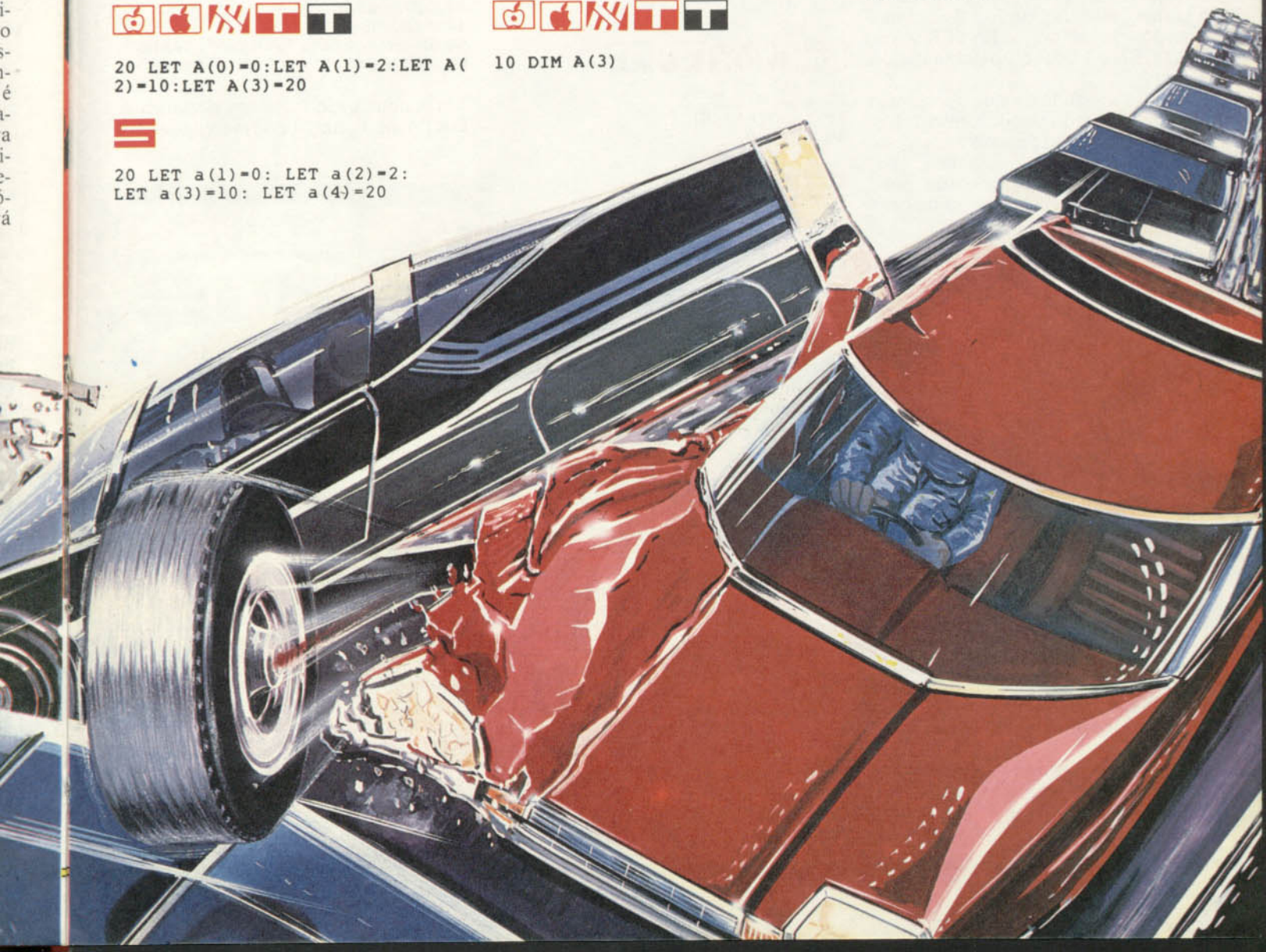
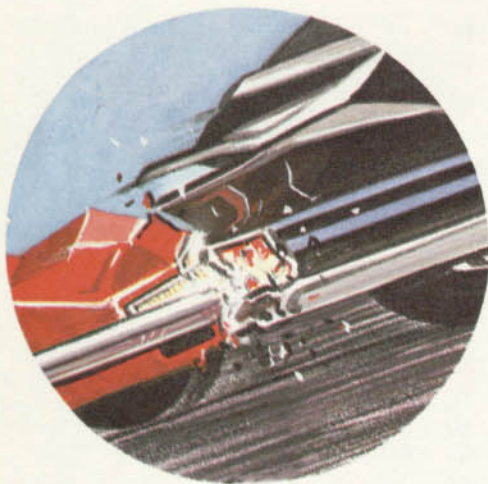
```
20 LET A(1)=0
30 LET A(2)=2
40 LET A(3)=10
50 LET A(4)=20
```

Até agora, não houve economia de tempo ou de espaço na memória. Ao contrário, teria sido bem mais rápido utilizar um simples comando **LET**. Mas veja em seguida o que acontece quando é necessário armazenar valores que são diferentes, cada vez que se roda o programa:

```
10 DIM A(3)
```

```
20 PRINT"QUAIS .SAO OS VALORES?"
30 INPUT A(0),A(1),A(2),A(3)
```

```
10 DIM a(4)
20 PRINT "Quais sao os valore
s?"
30 INPUT a(1),a(2),a(3),a(4)
```

```
10 DIM A(4)
20 PRINT "QUAIS SAO
OS VALORḞS?"
```

```
30 INPUT A(1)
40 INPUT A(2)
50 INPUT A(3)
60 INPUT A(4)
```

O computador pedirá o valor de cada variável, sempre que você rodar o programa. Bastará, então, digitar um número (seguido por <**ENTER**> ou <**RETURN**>) quando o computador solicitar.

Assim, na medida em que o programa tiver um número maior de elementos, o tempo economizado passará a compensar o trabalho inicial. Suponha, por exemplo, que você queira entrar uma centena de números. Um programa bem simples será o suficiente:

```
10 DIM A(99)
20 FOR N=0 TO 99
30 INPUT A(N)
40 NEXT N
```

No ZX-81 digite em letras maiúsculas.

```
10 DIM a(100)
20 FOR n=1 TO 100
30 INPUT a(n)
40 NEXT n
```

CONJUNTO DE NOMES

O tipo de conjunto descrito até aqui permite apenas o armazenamento de números. Mas também é possível utilizar conjuntos para armazenar cadeias alfanuméricas.

Se você quiser processar tanto nomes quanto números, por exemplo, precisará definir dois conjuntos separadamente: um para os nomes e outro para os números. O dimensionamento do conjunto específico para nomes é assim:

```
10 DIM A$(5)
```

Nos micros da linha Sinclair (ZX-81 e Spectrum) é necessário ainda especificar o comprimento máximo que a cadeia alfanumérica poderá ter. Assim, se o maior nome da lista tiver dez caracteres, os conjuntos de nomes serão dimensionados da seguinte maneira:

No ZX-81 digite em letras maiúsculas.

```
10 DIM a$(6,10)
```

Em cada caso o computador reservou espaço na memória para seis cadeias alfanuméricas (nomes, rótulos, etc.). Para definir o laço de entrada desses nomes, digite:

```
20 FOR N=0 TO 5
30 INPUT A$(N)
40 NEXT N
```

```
20 FOR n=1 TO 6
30 INPUT a$(n)
40 NEXT n
```

Rode o programa e entre os nomes à medida que forem pedidos na tela, seguindo-os por <**ENTER**> ou <**RETURN**>. Se você quiser verificar, logo após digitar cada nome, se o armazenamento foi correto, adicione esta linha ao programa acima e rode-o novamente:

```
35 PRINT A$(N)
```

```
35 PRINT a$(n)
```

Não é difícil digitar seis nomes, mesmo que sejam longos. E, ainda que o número de nomes chegue a cem, o trabalho será rápido. Imagine, por exemplo, que você esteja realizando uma pesquisa sobre o campeonato de Fórmula 1 e deseja armazenar os nomes das curvas e o número de batidas que ocorreram em cada uma delas durante uma temporada de corridas. Não será improvável que precise entrar algumas dezenas de curvas, ou até mais, dependendo do caso. Mas o princípio é o mesmo que o utilizado no programa a seguir, com apenas seis nomes. Este programa faz com que o computador leia a série de nomes de curvas do autódromo de Jacarepaguá, armazenados em declarações **DATA** (os micros da linha ZX-81 não dispõem desse recurso):

```
10 DIM A$(5)
20 FOR N=0 TO 5
30 READ A$(N)
40 NEXT N
50 DATA NORTE,NONATO,PACE
60 DATA UM,VITORIA,LAGOA
```

```
10 DIM a$(6,11)
20 FOR n=1 TO 6
30 READ a$(n)
40 NEXT n
50 DATA"NORTE","NONATO","PACE"
60 DATA"UM","VITORIA","LAGOA"
```

Os nomes são lidos nas declarações **DATA** na linha 50 e armazenados em

um conjunto, na linha 10. Assim, cada vez que o programa é rodado, os mesmos nomes entram automaticamente. No caso de existirem cem nomes, em vez de seis, modifique a linha 10 para **DIM A$(99)** e a linha 20 para **FOR N = 0 to 99** (para os micros Apple, TRS-80, TRS-Color e MSX), ou **DIM A$ (100, 11)** e **FOR N = 1 TO 100** (para o Spectrum). Em seguida, você poderá acrescentar os 94 nomes restantes às declarações **DATA**, a partir da linha 50.

O próximo passo será a definição do conjunto que conterá o número de acidentes em cada curva.

```
60 DIM A(5)
```

```
70 FOR N=0 TO 5
80 READ A(N)
90 NEXT N
100 DATA 0,2,5,1,3,6
```

```
 60 DIM a(6)
 70 FOR n=1 TO 6
 80 READ a(n)
 90 NEXT n
100 DATA 0,2,5,1,3,6
```

É possível ignorar o elemento zero de um conjunto, nos computadores Apple, MSX, TRS-80 e TRS-Color, se for mais conveniente. Assim, no último exemplo, podemos definir **DIM A$ (6)** e numerar os itens de 1 a 6. Isso significa que **A$(0)** fica vazio, pois é mais usual contar a partir do 1 do que 0.

UTILIZAÇÃO DOS CONJUNTOS

Agora o computador já "sabe" quantos acidentes ocorreram em cada curva. Mas como essa informação pode ser manipulada?

Provavelmente, você gostaria que, antes de mais nada, o computador imprimisse uma lista com todas as curvas e o número dos acidentes registrados em cada uma — somente para checar se os dados foram digitados corretamente. Adicione as linhas seguintes ao programa e rode-o de novo.

```
210 FOR N=0 TO 5
220 PRINT A$(N),A(N)
230 NEXT N
```

```
210 FOR n=1 TO 6
220 PRINT a$(n),a(n)
230 NEXT n
```

As linhas 210 e 230 percorrem a lista, enquanto a linha 220 imprime os nomes e os números armazenados nos conjuntos correspondentes. Se algum erro de digitação foi cometido, você terá a oportunidade de localizá-lo.

Você pode, também, querer analisar os resultados contidos nos conjuntos. Por exemplo, para o levantamento das corridas, seria interessante obter respostas para questões tais como: "quantas colisões ocorreram ao todo?"; "quais são as curvas mais seguras?". As linhas adicionais do programa, destinadas a identificar o total de acidentes, são as seguintes:

```
300 CLS
310 LET TL=0
320 FOR N=0 TO 5
330 LET TL=TL+A(N)
340 NEXT N
350 PRINT "NUMERO TOTAL DE ACID
ENTES ";TL
```

Substitua no programa anterior o comando **CLS** por **HOME**, na linha 300.

```
300 CLS
310 LET total=0
320 FOR n=1 TO 6
330 LET total=total+a(n)
340 NEXT n
350 PRINT "Numero total de aci
dentes: ";total
```

ANÁLISE DA INFORMAÇÃO

Imagine o quanto o programa acima seria útil se existissem cem curvas, em vez de apenas seis.

E, levando em conta que a informação não apenas pode ser facilmente armazenada em um conjunto, como também pode ser analisada com a mesma facilidade, você entenderá por que os conjuntos constituem um instrumento tão poderoso — para qualquer coisa, desde orçamento doméstico até finanças internacionais.

Digite estas linhas extras para examinar um exemplo de análise:

```
400 FOR N=0 TO 5
410 IF A(N)>3 THEN PRINT A$(N),
A(N)
420 NEXT N
```

```
400 FOR n=1 TO 6
410 IF a(n)>3 THEN  PRINT a$(n
 ),a(n)
420 NEXT n
```

Esta parte do programa mostra na tela uma lista das curvas nas quais ocorreram mais de três acidentes. Em nosso exemplo de seis nomes, elas são as curvas PACE e LAGOA. Se você substituir o número 3 pelo 5, na linha 410, e rodar o programa, apenas LAGOA será impresso. Para qualquer número maior do que 6 — nesse caso, o maior valor — nada será impresso.

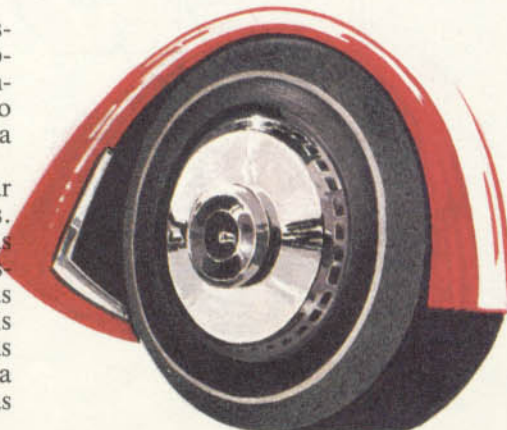
A informação armazenada nos conjuntos poderia ser, para dar um segundo exemplo, uma lista de famílias de uma cidade, juntamente com estatísticas como o número de crianças, os rendimentos mensais e o número de carros.

Uma vez que esses elementos tenham sido entrados, eles poderão ser classificados em grupos e analisados de inúmeras maneiras. Por exemplo, suponha que queiramos listar os dados de todas as famílias com sobrenome começando com P. Para ter uma idéia de como isso funciona, acrescente estas linhas ao programa:

```
600 FORN=0TO5
620 IFLEFT$(A$(N),1)="P"THENPRI
NTA$(N)
630 NEXT
```

```
600 FOR n=1 TO 6
620 IF a$(n,1)="P" THEN  PRINT
a$(n)
630 NEXT n
```

As linhas 600 e 630 definem um laço, para examinar cada elemento do conjunto de nomes. À medida que isso ocorre, a linha 620 checa o primeiro caractere. Se ele for P, um nome inteiro aparecerá.

Nesse caso, a curva chamada PACE será impressa, porque é a única que começa com P. No exemplo do conjunto de família, poderíamos escrever uma linha do programa que listasse todas as famílias 'SILVA', todas as que tivessem mais de um carro, e assim por diante. Outra possibilidade é o cruzamento de informações — por exemplo, quais as pessoas cujos nomes começam por 'A', que moram em determinada rua, no número 21, e cujo cachorro é um fila brasileiro!

TRADUÇÃO MANUAL DO ASSEMBLY

Rápidos e eficientes, os programas em código de máquina têm, contudo, uma desvantagem: é extremamente difícil escrevê-los e depurá-los. Freqüentemente, de fato, eles parecem apenas uma seqüência de números sem sentido, pois instruções, dados e endereços são todos especificados como números hexadecimais.

A solução não está em deixar de escrever programas em códigos de máquina. A maioria deles é escrita inicialmente em linguagem Assembly e depois traduzida para código de máquina. Normalmente, isto é feito por um outro programa, chamado Assembler (ou seja, montador). Porém, se você não dispuser de um Assembler — ou se quiser aprofundar seus conhecimentos — poderá fazer essa tradução manualmente e digitar os códigos hexadecimais resultantes na memória do computador, entrando o programa monitor ou o interpretador BASIC (comandos **PEEK** e **POKE**).

A LINGUAGEM ASSEMBLY

Você terá, assim, que aprender a linguagem Assembly, que está longe de ser tão difícil quanto os códigos de máquina. Os mnemônicos, que representam os códigos de operações da máquina, praticamente dispensam explicações, pois são abreviaturas muito claras dos comandos correspondentes (mnemônico significa que lembra algo). Os dados e endereços são todos numéricos, assim como os códigos de máquina. Mas, usando códigos alfabéticos para escrever os comandos, a seqüência pura de números será rompida, e ficará muito mais fácil para o programador entender o programa resultante.

Ao se programar em linguagem Assembly, portanto, basta consultar em uma tabela os códigos mnemônicos referentes ao microprocessador utilizado pelo seu computador.

O Sinclair Spectrum, o ZX-81, o TRS-80 e o MSX empregam o microprocessador Zilog Z-80, enquanto o TRS-Color usa o Motorola 6809, e o Apple II e o TK-2000 utilizam o 6502. O resultado da consulta à tabela é o código de operação numérico correspondente. Coloque-o no lugar correto, e a tradução estará praticamente feita.

Há algo mais a ser lembrado. Nos computadores das linhas Sinclair, Apple II, TRS-80 e MSX, não se pode esquecer de trocar a ordem dos bytes constantes de um endereço de memória, do programa, ou de outros dados que precisam ser representados por dois bytes (dezesseis bits). A razão disso é que essas máquinas armazenam números no formato de byte-baixo/byte-alto. O TRS-Color utiliza um formato inverso, ou seja, de byte-alto/byte-baixo, de modo que você poderá deixar os dados ou endereços de dezesseis bytes em sua ordem normal.

OS MNEMÔNICOS

Os mnemônicos — assim como os códigos de máquina em hexa que eles representam — servem para manipular o conteúdo de um registro, definir o valor de um sinalizador ou passar de um ponto a outro no programa. Estas são, basicamente, as únicas coisas que as instruções em código de máquina são capazes de realizar. Um exemplo de mnemônico é **LDA**, que significa **L**oa**D** **A**ccumulator (*carregar o acumulador*).

Se quisermos utilizar esse recurso, deveremos começar por aprender a traduzir os mnemônicos da linguagem Assembly para códigos hexadecimais.

ENDEREÇAMENTO

Contudo, traduzir programas em linguagem Assembly para código de máquina não é tão fácil quanto parece. Mesmo uma instrução simples como **LDA** pode ser traduzida para cinco a quinze códigos de operações diferentes, dependendo da máquina.

Os diversos códigos de operações dependem do tipo de endereçamento que está sendo utilizado, ou seja, das formas de acesso à memória RAM que são utilizadas pelo microprocessador.

Veremos a seguir como realizar a tradução correta de um programa em linguagem Assembly para os vários tipos de computador. Na próxima lição, aprenderemos a montar rotinas muito úteis em linguagem Assembly.

O tipo mais simples de endereçamento no ZX-81 é o chamado *endereçamen-*

Montar programas em código de máquina pode ser um risco para seu equilíbrio mental! Uma alternativa mais saudável é escrevê-los em Assembly e depois traduzi-los para hexadecimal.

to imediato, no qual um valor numérico é usado como argumento do código de operação:

LD A,4

significa: "carregar (**LOAD**) o acumulador (**A**) com o número 4"; essa instrução é traduzida para o código **3E04** em linguagem de máquina.

Já no *endereçamento direto* é fornecido um endereço onde o dado pode ser encontrado, ao invés do próprio dado. Por exemplo:

LD A,(0E2D)

significa: "carregar o conteúdo da locação **0E2D** no registro **A**". Isto se traduz para **3A 2D 0E** (observe que os dois bytes correspondentes ao endereço foram trocados).

O endereçamento direto também funciona de modo inverso:

LD (0E2D),A

significa: "carregar o conteúdo do registro **A** na locação de memória **0E2D**". Isto é traduzido para **32 2D 0E** (novamente os dois bytes foram intercambiados). Um terceiro tipo é o *endereçamento indireto*. Este diz à máquina onde encontrar o endereço do dado necessário. Por exemplo, a instrução:

LD A, (HL)

significa: "carregar o registro **A** com o dado pertencente ao endereço contido no par de registros **HL**".

Em outras palavras, o microprocessador examina o registro **HL** e usa esse número como o endereço da locação de memória onde está o byte a ser carregado no acumulador.

Esse comando funciona também na direção contrária:

LD (HL),A

significa: "carregar o conteúdo do registro **A** na locação de memória cujo endereço se encontra no par de registros **HL**".

Existe um tipo especial de endereçamento indireto que é chamado de *endereçamento indexado*. Aqui, um dos dois registros de indexação — **IX** e **IY** — e o próprio endereço a ser utilizado são fornecidos por um valor de deslocamento, o qual é acrescentado aos conteúdos dos registros **IX** ou **IY**.

Uma instrução típica poderia ser:

LD A, (IX+2F)

Observe que o deslocamento é de apenas um byte.

Os dados também podem ser transferidos de um registro para outro. Isso é chamado de *endereçamento de registro a registro*, cuja instrução correspondente é escrita assim:

LD D,B

que significa: "carregar o conteúdo do registro **B** no registro **D**".

O *endereçamento relativo* é empregado apenas com comando de desvio. Ele diz ao computador quantos bytes este deve saltar para a frente ou para trás. Por exemplo:

JRNZ 0FC

O mnemônico **JRNZ** (*Jump Relative on Non-Zero*) significa: "saltar **FC** bytes em relação à posição atual, se o resultado da última operação efetuada for diferente de zero", ou seja, se o sinalizador de zero no registro de sinalizadores (flag register) não for igual a 1. O **FC** diz para onde saltar.

FC é −4 em complemento de dois. Assim, se o sinalizador de zero não estiver ligado, o microprocessador salta quatro bytes para trás no programa, contados a partir do final da instrução **JRNZ 0FC**, e esta é traduzida para **20 FC** em código de máquina. Assim, o microprocessador salta de fato para a instrução que apareceu dois bytes antes dela.

Na realidade, raramente o endereçamento relativo é utilizado em linguagem Assembly. Normalmente os saltos são indicados por *rótulos* (*labels*). Estas são palavras inventadas pelo programador, que cumprem a função de marcadores — tal como **INICIO** — destinados a assinalar o começo de determinadas porções de um programa.

Esse marcador aparece na frente da instrução para onde será realizado o salto, ou ainda, logo após a instrução de salto. Por exemplo, a linha inicial poderia ser:

INICIO LD A,07

e em algum outro ponto do programa apareceria a instrução de salto:

DJNZ INICIO

O **DJNZ** (*Decrement and Jump on Non-Zero*) significa: "reduzir de 1 o valor do registro **B** e saltar para a instrução logo a seguir, onde o rótulo **INICIO** estiver colocado, no caso de o sinalizador de zero não estar ligado". Ao traduzir manualmente para código de máquina, você precisará elaborar os saltos relativos por sua própria conta.

A forma mais simples de endereçamento no microprocessador 6502 é o *endereçamento implícito*. Na verdade, ele não é propriamente um endereçamento. Por exemplo:

CLC

significa: "limpar o sinalizador de transporte (*Clear Carry Flag*)". Não é necessário nenhum tipo de endereço como argumento. A ação é executada no sinalizador de transporte, cujo endereço está implícito na instrução.

No endereçamento imediato o dado segue diretamente a instrução. Por exemplo:

LDA #&04

Essa operação carrega o acumulador com o número 4. O comando **LDA** é traduzido, na tabela de códigos de operações do 6502, para **A9** quando esta modalidade de endereçamento é utilizada. Assim, a instrução completa passa a ser **A904**.

Já no *endereçamento absoluto*, o endereço completo de uma locação de memória segue o mnemônico. Por exemplo:

LDA & 1122

significa: "carregar o acumulador com o dado armazenado na locação de memória **1122**". Essa forma também é conhecida como endereçamento direto.

A tradução para o **LDA** com endereçamento absoluto é o código hexadecimal **AD**. No exemplo acima, a instrução completa é traduzida para **AD 2211**. Observe que essa operação tem três bytes e que os dois bytes do endereço são trocados de ordem, quando se traduz da linguagem Assembly para o Apple.

Na página 0 — isto é, de 0000 a 00FF — não é preciso especificar o primeiro byte de endereço, mas deve-se utilizar um código de operação especial de página 0 (chamado de código de operação de endereço curto), o qual diz ao computador para olhar para um endereço de um byte.

O código de operação para **LDA** na modalidade de endereçamento de página 0 é A5. Assim, uma instrução como:

LDA &7F

será traduzida para **A57F**.

Com o endereço absoluto e o endereço de página 0, é possível utilizar o *endereçamento indexado*, no qual o conteúdo de um dos registros de indexação —**X** e **Y**— é acrescentado ao endereço dado com a instrução, para fornecer um segundo endereço que será utilizado. Por exemplo:

LDA &1122,X

Suponha que o conteúdo de registro **X** seja 33. Acrescentando-se 33 a 1122, obtém-se 1155; o acumulador é, então, carregado com o dado que se encontra na locação de memória 1155.

Ambos os registros **X** e **Y** podem ser empregados para indexar tanto o endereçamento absoluto quanto o de página 0.

Mas note que, se a soma do conteúdo do registro **X** com o endereço da página 0 for maior do que FF e, em virtude disso, cair na página 1, o byte mais significativo será ignorado. No endereçamento de página zero, indexado ou qualquer outro, o endereço utilizado sempre estará na página 0.

Ao usar endereçamento indexado deve-se consultar o código mnemônico em Assembly, correspondente à página 0X, página 0Y, absoluto X ou absoluto Y.

Também é possível endereçar indiretamente uma locação de memória, utilizando-se o endereçamento indireto. Nesta forma, o código de operação é seguido por um endereço entre parênteses. Isso significa que o microprocessador encontrará nesta locação um segundo endereço, o qual será empregado para acessar o dado. Por exemplo:

JMP (&1530)

significa: salte para o endereço dado na locação de memória 1530. Mas como qualquer locação de memória pode reter apenas um byte, e como são necessários dois bytes para compor um endereço, o microprocessador olha para 1530 e 1531. A primeira locação contém o byte menos significativo, e a segunda, o byte mais significativo, de acordo com a convenção utilizada para esse microprocessador. Assim, se a locação de memória 1530 contém 2F, e a locação de memória 1531 contém 13, o microprocessador saltará para a locação de memória 132F.

Existem também duas maneiras de indexar endereços indiretos. Com o registro **X** você poderá acrescentar um deslocamento ao primeiro endereço (o endereço fornecido na instrução). O procedimento é chamado de *endereçamento indireto pré-indexado*. Alternativamente, você poderá acrescentar um deslocamento do registro **Y** para o segundo endereço (o endereço nas locações de memória fornecido na instrução original). Isso é chamado de *endereçamento indireto pós-indexado*.

Esta é a aparência de instruções deste tipo, em linguagem Assembly:

LDA (&1122,X) e LDA (&1122),Y

O primeiro código usa o endereçamento pré-indexado, e o segundo, o pós-indexado. Para se consultar os códigos hexadecimais correspondentes, deve-se procurar o **LDA** sob **(indireto,X)** e **(in-**

direto),Y. Isso fornecerá **A1** e **B1** respectivamente. Assim, as duas instruções acima seriam traduzidas para **A1 2211** e **B1 22 11**.

As instruções de desvio são saltos condicionais. Por exemplo:

BEQ

significa: desviar se for igual a (**B**ranch if **EQ**ual) — isto é, se o sinalizador zero estiver igual a 1. As instruções de desvio podem utilizar endereçamento relativo em alguns tipos de Assembler.

BEQ #&04

significa: salte quatro bytes para a frente a partir do início da próxima instrução, se o sinalizador zero estiver igual a 1.

Por meio de rótulos, indica-se ao microprocessador o ponto do programa para onde o desvio deve ser realizado. A primeira instrução a partir deste ponto é simplesmente precedida pelo rótulo. Por exemplo:

INICIO LDA &04

ao passo que a instrução que produz o desvio terá:

BEQ INICIO

O Assembler se encarregará, então, de calcular o desvio relativo. No entanto, se você está traduzindo manualmente uma listagem em Assembly, a responsabilidade pelo cálculo será sua. Código de máquina não emprega rótulos, apenas números. Os microprocessadores 6510 e 6502 comportam ainda mais um tipo de endereçamento — o *endereçamento por acumulador* —, usado principalmente com instruções de deslocamento e rotação de bits. Por exemplo:

ASL A

significa: desloque o acumulador de um bit (**A**ccumulator **S**hift **L**eft). É possível utilizar esta instrução, também, com outras locações de memória, e não apenas com o acumulador. Neste caso, basta substituir o A, no exemplo acima, pelo endereço de memória cujo conteúdo você quer deslocar. Esta locação pode ser indexada com o registro **X**, apenas.

A instrução tem o seguinte efeito: o bit menos significante da memória é zerado, o bit mais significante é colocado no registro indicador de transporte (*carry flag*), e os bits restantes são deslocados uma posição à esquerda. Da mesma forma, existe o **ASR** (**A**ccumulator **S**hift **R**ight), para efetuar deslocamentos de bits à direita.

T

O microprocessador Motorola 6809, utilizado nos computadores compatíveis com a linha TRS-Color, tem um tipo de endereçamento implícito, também chamado *endereçamento de registros*.

Esta expressão refere-se às instruções da linguagem Assembler que não precisam ser acompanhadas de um endereço pois operam em um registro interno do microprocessador, que é estipulado pela própria instrução. Por exemplo:

DECA

significa: diminua de 1 o registro **A** (Decrement Accumulator). O código hexadecimal correspondente, que pode ser encontrado na tabela de conversão do manual do microprocessador, é 4A.

No endereçamento imediato, o dado a ser utilizado é o próprio argumento da instrução. Por exemplo:

ADDB #$7

significa: adicione 7 ao registro **B**. O código para ADD em modo imediato é CB, em hexadecimal, de modo que a instrução completa acima é traduzida para CB 07.

Existe ainda outra forma de endereçamento imediato, na qual se usa um segundo byte, chamado *pós-byte*. Neste caso, a instrução assume a forma:

TFR A,B

que significa: transferir o conteúdo do registro **A** para o registro **B**.

Para traduzir essa instrução para código de máquina, deve-se consultar o código TFR na seção de endereçamento imediato da tabela. Isto nos dá o código IF. A seguir, o pós-byte é avaliado um *nibble* (um grupo de quatro bits) de cada vez.

Em geral, para usar esta instrução com o 6809, atribui-se um dígito hexadecimal para cada registro. Dois desses dígitos são unidos para formar o pós-byte. Por exemplo, o registro A recebe o valor de 8, e o registro B, o valor de 9. De modo que:

TFR A,B

é traduzido para IF 89. Se, ao contrário, a instrução tivesse sido:

TFR B,A

a instrução correspondente em código de máquina seria IF 98.

No endereçamento absoluto, ou *ampliado*, a instrução é acompanhada do endereço de dezesseis bits da locação de memória a ser utilizada. Por exemplo:

STA $7530

significa: armazene o conteúdo do acumulador da memória &H7530 (Store Accumulator). A tradução, no caso, seria B7 75 30 (note que os dois últimos bytes, correspondentes ao endereço, são mantidos na ordem direta, como é obrigatório para a maior parte dos outros tipos de microcomputador).

A instrução é um pouco complicada, pois envolve três bytes. Usando-se endereçamento direto, porém, podemos reduzi-la para uma instrução de dois bytes. O microprocessador 6809 tem um registro de página direta, que armazena o byte mais significativo de um endereço. Assim, todas as memórias naquela página podem ser endereçadas usando-se apenas um byte, no caso o menos significante.

Esse recurso funciona da seguinte

maneira: o registro de página direta é carregado usando-se a instrução **TFR**, mencionada acima, ou a instrução **EXG** (**EX**chan**G**e), que troca os conteúdos de dois registros entre si.

EXG A,DP

Na instrução acima, pede-se para intercambiar os registros A e página direta (Direct Page). A instrução não apenas define a página direta com o que foi carregado em A com um comando **LDA** prévio; ela também armazena o número de página direta previamente retido por **DP**, de volta para o registro **A**. Dali esse número pode ser jogado para uma locação de memória qualquer, usando a instrução **STA**.

O byte mais significante da página direta — 75, neste exemplo — está agora definido, e o byte menos significante pode ser dado com a instrução. Neste caso, para fazer a tradução manual, é necessário consultar a instrução **STA** na tabela de códigos, sob a seção de endereçamento indireto. O resultado é 97, em hexadecimal. Assim, a instrução completa do exemplo será 9730.

Obviamente, não vale a pena repetir todo este procedimento cada vez que se quiser armazenar dados em uma locação de memória. Definir o endereço-base da página direta toma muito mais tempo do que o economizado com a redução da instrução de três bytes para dois bytes. Mas muitas vezes é suficiente definir a página direta no começo de um segmento de programa e, daí em diante, armazenar todos os dados na mesma página.

Endereçamento indireto ocorre quando o microprocessador precisa examinar uma memória indicada por um endereço que, por sua vez, contém um segundo endereço, no qual estão os dados que serão carregados no acumulador. A forma de notação utilizada é entre colchetes:

LDA [$FEEE]

Por meio desta instrução, pede-se ao microcomputador para recuperar um endereço, nas locações de memória FEEE e FFFF (o endereço completo, com dois bytes).

Prosseguindo, a instrução carrega no acumulador A o conteúdo na locação com este segundo endereço.

O endereço indireto também pode ser realizado com os registros **U**, **S**, **X** e **Y**, bem como com o cortador de programa.

Em todos esses casos de endereçamento indireto, é necessário consultar o mnemônico da instrução na tabela, sob a seção de endereçamento indexado, para se obter o código correspondente. Em seguida, deve-se consultar o pós-byte.

No exemplo dado acima, consulta-se **LDA** sob a seção de endereçamento indexado, e se obtém A6. Depois, consulta-se o endereço [mmmm], que indica um endereço geral de dezesseis bytes, na tabela de pós-bytes. Isto dá 9 F (ou BF, ou DF ou FF, pois os pós-bytes são repetidos para cada registro: como [mmmm] é independente de qualquer registro, aparece quatro vezes).

Assim a instrução completa:

LDA [$FFFE]

é traduzida para A6 9F FF FE.

Usando os registros **U**, **S**, **X** e **Y**, bem como o contador de programa, pode-se estipular tanto o endereçamento não-indireto (sem colchetes), quanto o endereçamento direto (com colchetes), usando-se *deslocamentos* (*offsets*). Os deslocamentos podem ser constantes — decimais, hexadecimais de oito ou dezesseis bits — ou o conteúdo de outros registros. Este conteúdo é somado ou subtraído do conteúdo-base de um dos cinco endereços indexáveis.

LDA 0 ,X

significa: carregue o registro acumulador com dados na locação de memória, cujo endereço está armazenado no registro **X**. Nesse exemplo, o deslocamento é zero. Outros exemplos:

LDA 1,X

significa: carregar o acumulador com o endereço em **X**, somado de 1.

LDA –16,Y

significa: carregar o acumulador com o endereço em **Y**, diminuído de 16.

LDA ($10, X)

é a versão para endereçamento indexado. Ele carrega o acumulador com o conteúdo da locação de memória cujo endereço está indicado em **X**, e que é adicionado de 10 (em hexadecimal).

Em todos estes casos, o código operacional para **LDA** pode ser encontrado na seção de endereçamento indexado, e o pós-byte apropriado precisa ser adicionado, a partir do que for encontrado na tabela de pós-bytes. Deslocamentos de oito bits e endereços de dezesseis bits seguem-se a isto.

Desvios em linguagem Assembler são programados por meio das instruções **JMP** e **JSR**. Estes desvios podem ser definidos em termos de endereços finais de página direta, ampliados, ou, ainda, indexados, de modo que os códigos de máquina equivalentes se alteram de acordo. Desvios condicionais, entretanto, utilizam endereçamento relativo.

Com endereçamento relativo, informa-se à UCP quantos bytes o programa deve pular para a frente ou para trás em relação à instrução de desvio.

Existem dois tipos de desvios: um desvio ordinário de oito bits, que pula para até 127 bytes para a frente ou 128 bytes para trás, e um desvio longo de dezesseis bits, que pode abranger 32 767 bytes para trás.

BEQ $FA

pulará seis bytes para trás, a partir do início da instrução seguinte, se o indicador de zero (*zero flag*) estiver ligado. FA é –6 em complemento de dois. Alternativamente, temos o desvio longo:

LBEQ $0A00

que saltará 2560 bytes para a frente, se o indicador de zero estiver ligado.

Os desvios sempre utilizam endereçamento relativo — não há outro tipo de endereçamento disponível para eles. Porém, como muitas vezes é trabalhoso calcular o tamanho de um salto, procura-se utilizar rótulos.

Os rótulos são palavras (simbólicas) criadas pelo programador, que indicam ao microprocessador qual é o ponto do programa para onde o desvio deve ser realizado e, a partir deste ponto, a primeira instrução é simplesmente precedida pelo rótulo. Por exemplo:

INICIO DECA

ao passo que a instrução que produz o desvio terá:

BEQ INICIO

O Assembler se encarregará, então, de calcular o desvio relativo, que poderá ser para a frente, se o rótulo estiver depois da instrução de desvio, ou para trás, se estiver antes dela. Entretanto, se você está traduzindo manualmente uma listagem em Assembly, a responsabilidade pelo cálculo será sua, já que código de máquina não emprega rótulos, apenas números. Para fazer o cálculo, conta-se o número de bytes a partir do final da instrução de desvio até o começo da instrução para a qual se deseja saltar. Lembre-se que, para um salto longo, isto tomará dois bytes.

CONJUNTOS DE DUAS DIMENSÕES

- DEFINIÇÃO DE MATRIZES
- ENTRE DADOS EM UMA MATRIZ
- COMO PROGRAMAR MATRIZES
- RECUPERAÇÃO DE INFORMAÇÕES
- USOS PARA AS MATRIZES

Se você tem uma grande quantidade de dados inter-relacionados, a informação pode estar perdida entre eles. Coloque-os numa matriz e o computador extrairá exatamente a informação desejada.

Agora que você já conhece como um conjunto pode ser empregado para conservar informações em um programa (lição *Conjuntos: Caixas de Informação*, página 192), podemos passar ao estudo de matrizes. Matriz (ou matrix) é todo conjunto *bidimensional* que serve de meio para armazenar grupos de dados *inter-relacionados*.

Um conjunto bidimensional é, graficamente, como uma pilha de gavetinhas, onde cada gaveta serve para armazenar um único dado. Esta é uma maneira muito conveniente de lidar com informações pois, para localizar uma delas, basta se referir a um elemento do conjunto, por meio dos seus números de linha e de coluna.

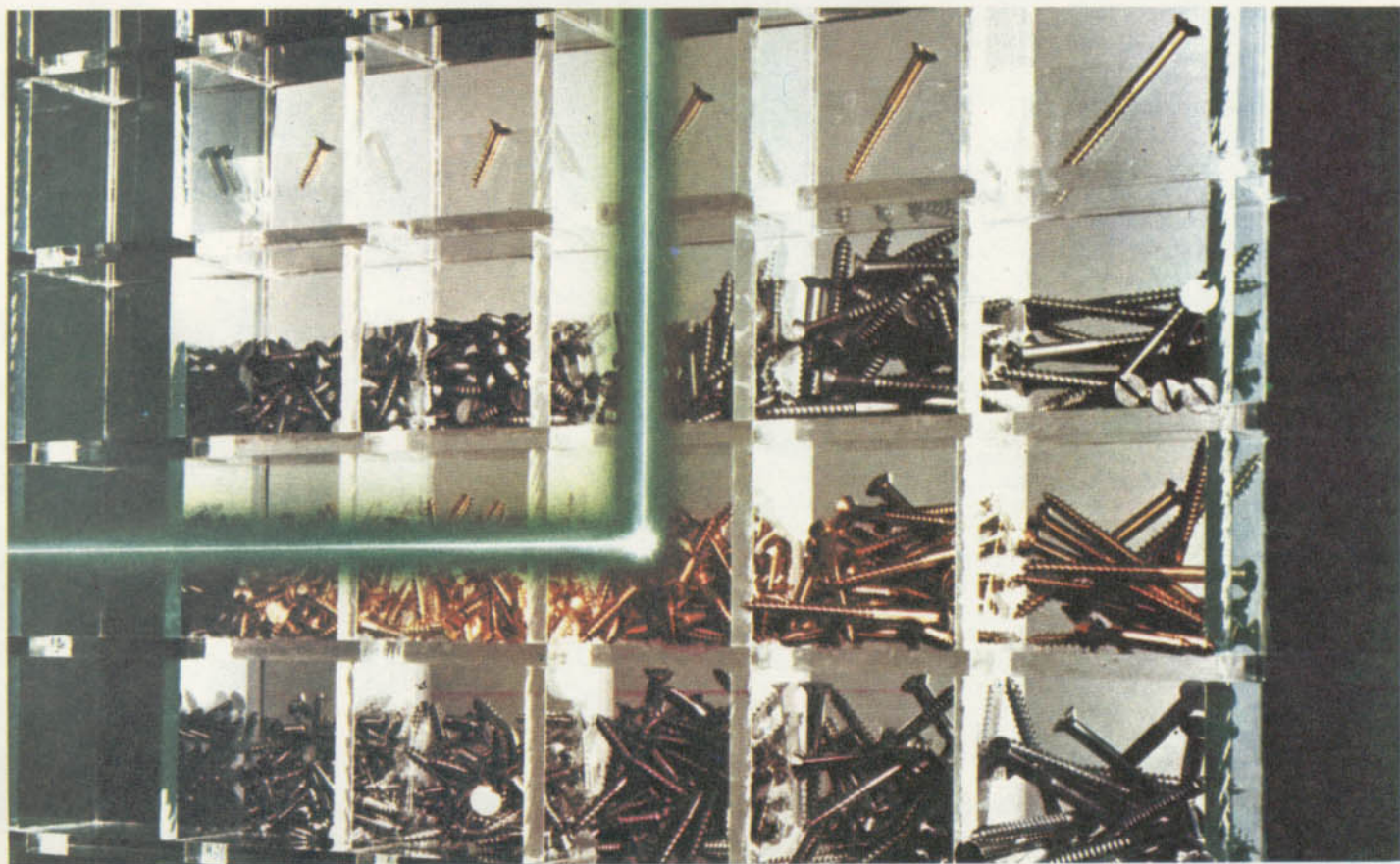
A matriz armazena itens reais, tais como os parafusos da ilustração abaixo, onde se pode encontrar imediatamente a caixa que contém parafusos de latão de 38 mm, ou para qualquer outra que se esteja procurando. Neste caso o dado da matriz é o número de parafusos em cada caixa.

A matriz pode também guardar informações abstratas, como o número de revendedores de automóveis, discriminados segundo a sua nacionalidade e marca dos carros vendidos.

Informações como esta são freqüentemente colecionadas como resultado de levantamentos, que são a base para o programa exposto neste artigo. Entretanto, a estrutura do programa é a mesma para qualquer matriz bidimensional, pouco importando o tipo de informação que ela contém.

Como exemplo, suponha que você seja um professor interessado em classificar os animais de estimação trazidos por um conjunto de crianças. Em primeiro lugar, seria preciso anotar o nome das crianças e descrever quantos animais de cada grupo elas possuem. A lista seria mais ou menos assim:

Sílvia: dois periquitos, um coelho.
João: um cachorro, quatro peixes.
Carlos: dois hamsters, um gato.

Recurso muito útil para armazenar dados, as matrizes servem também para classificar pequenos itens como parafusos.

Jaci: um cachorro, um gato, um hamster.
Beto: um rato, um coelho, dois peixes.

Entretanto, em uma lista como essa torna-se um tanto difícil extrair informações como "quantas pessoas possuem gatos?", ou "quem tem mais de quatro animais?" — e quanto maior for a lista, mais difícil será também obter respostas.

A melhor maneira de resolver o problema seria distribuir as informações em forma de tabela ou quadro, com o nome dos animais dispostos no alto e o das crianças à esquerda. Pois bem, é exatamente assim que funciona uma matriz bidimensional.

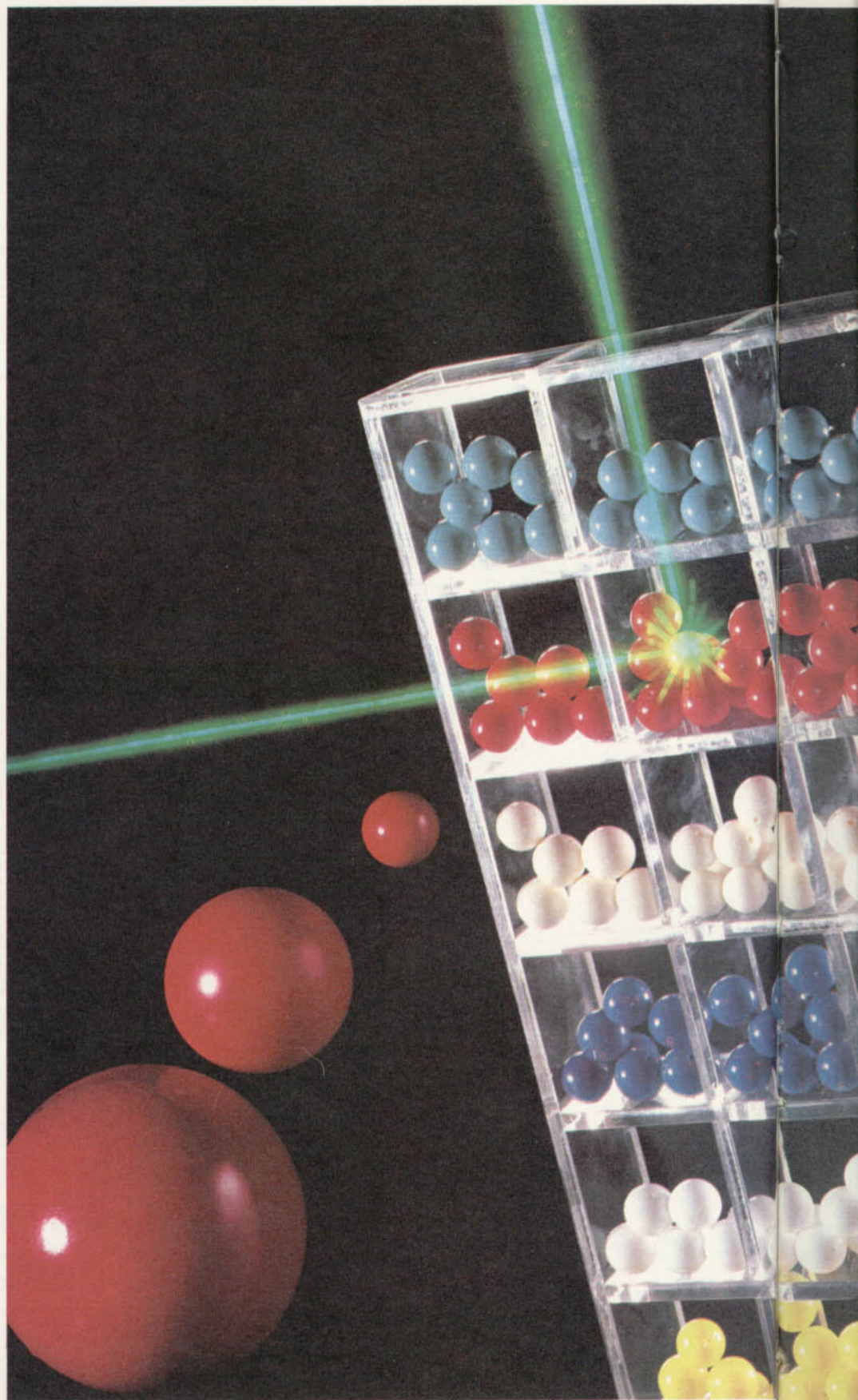
Em um exemplo pequeno como este é simples encontrar a resposta para qualquer pergunta sobre os dados. Entretanto, o computador trabalha com um infindável número de informações. Se você examinar todo o conjunto de dados para descobrir quem tem pelo menos um gato, ou se quiser saber o nome das pessoas que possuem ratos, verificará que o trabalho será tanto maior quanto mais ampla for a quantidade de dados. Por sorte, o computador pode cumprir essa tarefa em questão de segundos.

DEFINA A MATRIZ

Equacionado o problema, o passo seguinte será colocar a informação dentro do computador. Se você voltar às páginas 192 a 195 verá como definir e usar uma matriz unidimensional (também chamada de conjunto). Matrizes bidimensionais são muito similares, envolvendo apenas um pouco mais de trabalho para serem definidas.

Uma matriz unidimensional pode ser escrita na forma **A(N)**, onde **N** é o número dos seus elementos. Uma matriz bidimensional, por sua vez, é definida como **A(L,C)**, onde **L** é o número de linhas e **C** o número de colunas. Na verdade, pouco importa se você a definir de forma invertida, como **A(C,L)**, com as colunas antes das linhas, desde que seja coerente com a primeira ou a segunda definições.

Contudo, mesmo trabalhando com uma matriz bidimensional, você precisará definir duas matrizes unidimensionais que sirvam de cabeçalho para as colunas e para as linhas (no nosso caso, uma matriz conterá o nome das crian-

Assim como as bolinhas da ilustração, distribuídas ordenadamente segundo seu tamanho e cor, as informações devem ser armazenadas na memória de acordo com o grupo a que pertencem.

ças e a outra o nome dos animais). A matriz bidimensional conterá os dados.

Vamos chamar as duas primeiras matrizes de **PET$ (C)** e **CH$ (R)** e a matriz de dados de **N (R,C)**. Uma outra matriz, **PT (R)**, conterá o número total de animais em cada linha. No Spectrum, as matrizes são rotuladas de **p$ (c)**, **c$ (r)**, **n (r, c)** e **p (r)**, pois é permitido apenas um caractere para a formação dos nomes. O programa não funcionará no ZX-81. Eis um programa para dimensionar a matriz e ler os dados de um comando **DATA**.

```
100 C=7:R=5
110 DIM PET$(C),CH$(R),N(R,C),P
T(R)
120 FOR J=1TOC:READPET$(J):NEXT
130 FORJ=1TOR:READCH$(J):NEXT
140 FOR J=1TOR
150 FORK=1TOC
160 READ N(J,K):NEXT:NEXT
3000 DATA PERIQUITO,GATO,CACHOR
RO,PEIXE,RATO,HAMSTER,COELHO
3010 DATA SILVIA,JOAO,CARLOS,JA
CI,BETO
3020 DATA 2,0,0,0,0,0,1
3030 DATA 0,0,1,4,0,0,0
3040 DATA 0,1,0,0,0,2,0
3050 DATA 0,1,1,0,0,1,0
3060 DATA 0,0,0,2,1,0,1
```

```
100 LET c=7: LET r=5
110 DIM p$(c,7): DIM c$(r,5):
DIM n(r,c): DIM p(r)
120 FOR j=1 TO c: READ p$(j):
NEXT j
130 FOR j=1 TO r: READ c$(j):
NEXT j
140 FOR j=1 TO r
150 FOR k=1 TO c
160 READ n(j,k): NEXT k: NEXT
j
3000 DATA "PERIQUITO","GATO","C
ACHORRO","PEIXE","RATO","HAMSTE
R","COELHO"
3010 DATA "RODRIGO","RICARDO","
FERNANDA","BEATRIZ","JUNIOR"
3020 DATA 2,0,0,0,0,0,1
3030 DATA 0,0,1,4,0,0,0
3040 DATA 0,1,0,0,0,2,0
3050 DATA 0,1,1,0,0,1,0
3060 DATA 0,0,0,2,1,0,1
```

As variáveis **R** e **C** foram usadas para facilitar a adaptação do programa: afinal, é improvável que sua matriz tenha o mesmo número de linhas e de colunas que a que apresentamos aqui. Assim, tudo o que você tem a fazer é trocar os números na linha 100 e, logicamente, as informações nas linhas dos comandos **DATA**.

Note agora que os dados entram na matriz. Existe uma linha para o cabeçalho das colunas, outra para o cabeçalho das linhas e uma para cada linha da matriz principal. Note também que você deve entrar um item de dado para cada espaço na matriz, mesmo que ele seja zero, ou o computador responderá com uma mensagem de erro.

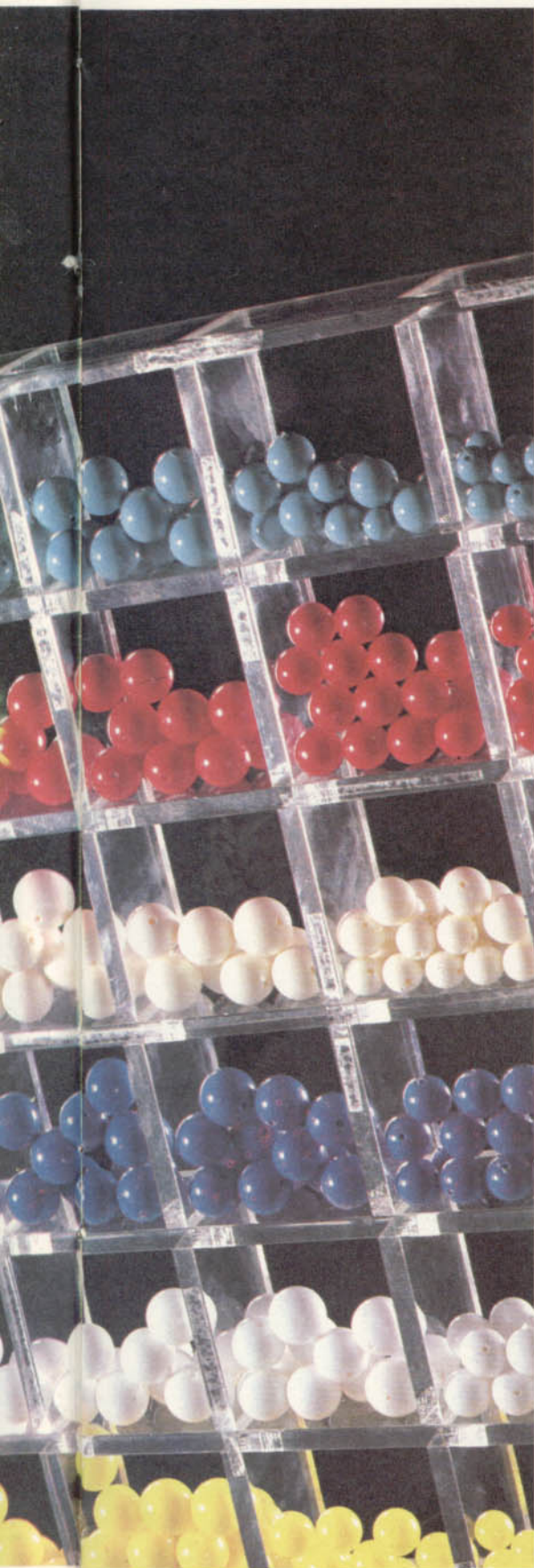
Agora que os dados estão dentro do computador, você deve decidir o que fazer com eles. Um de seus recursos é oferecer tantas opções quanto possível: afinal, é o computador que vai trabalhar, e não você.

O MENU

Será melhor descrever as opções em forma de menu. E para a pesquisa dos animais você provavelmente precisará do seguinte:

```
300 LET fo=0: CLS
310 PRINT ''"MENU"
320 PRINT '"1- LISTAR ANIMAIS"
330 PRINT "2- LISTAR CRIANCAS"
340 PRINT "3- INTRODUZIR TIPO
DE ANIMAL"
350 PRINT "4- INTRODUZIR NOME
DA CRIANCA"
360 PRINT "5- INTRODUZIR NUMER
O DE ANIMAIS"
370 PRINT "6- MOSTRAR A MATRIZ
"
380 PRINT ''"SELECIONE OPCAO"
390 INPUT a
395 IF a<1 OR a>6 THEN  GOTO
390
400 CLS
410 GOSUB (a*100+400)
420 GOTO 300
```

No TRS-80, multiplique por 2 os valores utilizados com o **PRINT@**.

```
300 FOUND=0:CLS
310 PRINT @43,"MENU"
320 PRINT @98,"1 LISTAR ANIMAIS
330 PRINT @130,"2 LISTAR CRIANC
AS
340 PRINT @162,"3 INTRODUZIR TI
PO DE ANIMAL
350 PRINT @194,"4 INTRODUZIR NO
ME DA CRIANCA"
360 PRINT @226,"5 INTRODUZIR NU
MERO DE ANIMAIS"
370 PRINT @258,"6 MOSTRAR A MAT
RIZ"
380 PRINT:PRINT"ESCOLHA A OPCAO
";
390 INPUT A
395 IF A<1 OR A>6 THEN 390
400 CLS
410 ON A GOSUB 500,600,700,800,
900,1000
420 GOTO 300
```

```
300 FO=0:CLS
310 LOCATE 17,1:PRINT"MENU"
320 LOCATE 7,4:PRINT"1. LISTAR
ANIMAIS"
330 LOCATE 7:PRINT"2. LISTAR CR
IANÇAS"
340 LOCATE 7:PRINT"3. INTRODUZI
R TIPO DE ANIMAL"
350 LOCATE 7:PRINT"4. INTRODUZI
R NOME DE CRIANÇA"
360 LOCATE 7:PRINT"5. INTRODUZI
R NÚMERO DE ANIMAIS"
370 LOCATE 7:PRINT"6. MOSTRAR A
 MATRIZ"
380 PRINT:PRINT:LOCATE 20:PRINT
"OPÇÃO";
390 INPUT A
395 IF A<1 OR A>6 THEN 390
400 CLS
410 ON A GOSUB 500,600,700,800,
900,1000
420 GOTO 300
```

```
300 FO = 0: HOME
310  HTAB 18: VTAB 2: PRINT "ME
NU"
320  VTAB 5: HTAB 8: PRINT "1.
LISTAR ANIMAIS"
330  HTAB 8: PRINT "2. LISTAR C
RIANCAS"
340  HTAB 8: PRINT "3. INTRODUZ
IR TIPO DE ANIMAL"
350  HTAB 8: PRINT "4. INTRODUZ
IR NOME DE CRIANCA"
360  HTAB 8: PRINT "5. INTRODUZ
IR NUMERO DE ANIMAIS"
370  HTAB 8: PRINT "6. MOSTRAR
A MATRIZ"
380  PRINT : PRINT : HTAB 20: P
RINT "OPCAO ";
390  INPUT A
395  IF A < 1 OR A > 6 THEN 390

400  HOME
410  ON A GOSUB 500,600,700,800
,900,1000
420  GOTO 300
```

ESCREVA AS SUB-ROTINAS

As sub-rotinas a seguir servirão para o início do trabalho, embora, mais tarde, você possa adicionar outras opções. A primeira imprime apenas a lista de animais, fazendo uma procura na matriz **PET$( )** ou **p$( )**. A segunda faz a mesma coisa para imprimir a lista de crianças.

```
499 REM ** OPCAO 1 **
500 PRINT '''LISTA DE ANIMAIS"
510 PRINT ''
520 FOR j=1 TO c
530 PRINT p$(j)
540 NEXT j
550 PRINT ''" PRESSIONE QUALQU
ER TECLA PARA   RETORNAR"
560 PAUSE 0
570 RETURN
599 REM ** OPCAO 2 **
600 PRINT '''LISTA DE CRIANCAS
"
610 PRINT ''
620 FOR j=1 TO r
630 PRINT c$(j)
640 NEXT j
650 PRINT ''"PRESSIONE QUALQUE
R TECLA PARA RETORNAR"
660 PAUSE 0
670 RETURN
```

```
499 REM ### OPCAO 1 ###
500 PRINTTAB(5);"LISTA DE ANIMA
IS"
510 PRINT:PRINT
520 FOR J=1TOC
530 PRINT PET$(J)
540 NEXT
550 PRINT:PRINT"PRESSIONE QUALQ
UER TECLA PARA RETORNAR"
560 IF INKEY$=""THEN560
570 RETURN
599 REM ### OPCAO 2 ###
600 PRINTTAB(5);"LISTA DE CRIAN
ÇAS"
610 PRINT:PRINT
620 FOR J=1TOR
630 PRINT CH$(J)
640 NEXT
650 PRINT:PRINT"PRESSIONE QUALQ
UER TECLA PARA RETORNAR"
660 IF INKEY$=""THEN660
670 RETURN
```

```
499  REM   *** OPCAO 1 ***
500  PRINT  TAB( 10)"LISTA DE A
NIMAIS"
510  PRINT : PRINT
520  FOR J = 1 TO C
530  PRINT PET$(J)
540  NEXT
550  PRINT : PRINT "PRESSIONE Q
UALQUER TECLA PARA RETORNAR";
560  GET A$
570  RETURN
599  REM  *** OPCAO 2 ***
600  PRINT  TAB( 10);"LISTA DE
CRIANCAS"
610  PRINT : PRINT
620  FOR J = 1 TO R
630  PRINT CH$(J)
640  NEXT
650  PRINT : PRINT "PRESSIONE Q
UALQUER TECLA PARA RETORNAR";
660  GET A$
670  RETURN
```

A opção 3 é a mais interessante. Se você digitar o nome de um animal, o computador imprimirá uma lista de crianças que têm no mínimo um deles, informando ainda quantos espécimes cada criança possui.

```
699 REM ** OPCAO 3 **
700 PRINT '''INTRODUZA TIPO DE
 ANIMAL"
705 DIM i$(7): INPUT  LINE i$
710 PRINT '"PESSOAS QUE TEM UM
 ";i$
720 FOR j=1 TO c
730 IF p$(j)=i$ THEN  LET fo=j
740 NEXT j
750 IF fo=0 THEN  PRINT " ANIM
AL NAO ENCONTRADO.  TENTE  OU"
RA VEZ.": GOTO 700
760 FOR j=1 TO r
```

A matriz mostra alguns revendedores de carros classificados por país e marca de automóvel.

```
770 IF n(j,fo)>0 THEN  PRINT c
$(j);" ";n(j,fo)
775 NEXT j
780 PRINT '"" PRESSIONE QUALQU
ER TECLA PARA  RETORNAR"
785 PAUSE 0
790 RETURN
```

```
699 REM ### OPCAO 3 ###
700 PRINTTAB(5);"DIGITE TIPO DE
 ANIMAL"
705 PRINT"LISTA DE TODOS QUE TE
M ";
710 INPUT P$
715 PRINT:PRINT
720 FORJ=1TOC
730 IFPET$(J)=P$THENFO=J
740 NEXT
750 IFFO=0THENPRINT"ANIMAL NÃO
ENCONTRADO. TENTE DE NOVO!":GOT
O700
760 FORJ=1TOR
770 IFN(J,FO)>0THENPRINTCH$(J);
" ";N(J,FO)
775 NEXT
780 PRINT:PRINT"PRESSIONE QUALQ
UER TECLA PARA RETORNAR";
785 IFINKEY$=""THEN785
790 RETURN
```

```
699   REM  *** OPCAO 3 ***
700   PRINT  TAB( 10);"DIGITE O
TIPO DE ANIMAL"
705   PRINT  TAB( 5);"LISTA DE T
ODOS QUE TEM ";
710   INPUT P$
715   PRINT : PRINT
720   FOR J = 1 TO C
730   IF PET$(J) = P$ THEN FO =
J
740   NEXT
750   IF FO = 0 THEN  PRINT "ANI
MAL NAO FOI ENCONTRADO. TENTE
 NOVAMENTE!": GOTO 700
760   FOR J = 1 TO R
770   IF N (J,FO) > 0 THEN  PRI
NT CH$(J);" ";N(J,FO)
775   NEXT
780   PRINT : PRINT "PRESSION
E QUALQUER TECLA PARA RETOR
NAR"
785   GET A$
790   RETURN
```

Esta rotina é mais complicada, de modo que vale a pena explicá-la com mais detalhes.

A linha 710 guarda nossa entrada na variável **P$** (**i$** no Spectrum). Digamos, por exemplo, que nossa entrada seja "GATO"; assim, **P$ = "GATO"** (ou **i$ = "GATO"**). As linhas 720 a 740 verificam se há algum **"GATO"** na lista de animais. Em caso positivo, a variável **FO** receberá o número da coluna em que "GATO" foi achado. No nosso caso, **FO=2** porque "GATO" está na coluna 2.

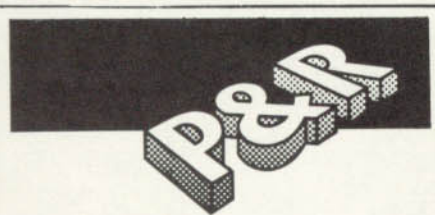
Se o computador atingir a linha 750 e **FO** for igual a 0 (ou seja, se seu valor for igual a 0 na linha 300), isso significa que o animal pedido não está na lista. Em conseqüência o programa recomeçará.

As linhas 760 a 775 percorrem cada elemento na coluna 2. Se algum elemento for maior que 0, isso significa que a pessoa nessa linha tem um gato e seu nome será impresso junto com a informação de quantos animais dessa espécie ela tem.

Verifique o programa e veja se você pode seguir o que o computador está fazendo a cada passo. A opção 4 permite

entrar o nome de uma pessoa para ver a lista de seus animais.

O programa funcionará de modo quase igual ao da última rotina. Desta vez, entretanto, o computador procurará por uma linha particular da matriz, e não por uma coluna, como acontecia no exemplo anterior.

```
799 REM ** OPCAO 4 **
800 PRINT '"INTRODUZA O NOME
DA CRIANCA"
805 DIM f$(5): INPUT  LINE f$
810 PRINT "ANIMAIS PERTENCENTE
S A ";f$
815 PRINT ''
820 FOR j=1 TO r
830 IF c$(j)=f$ THEN  LET fo=j
840 NEXT j
```

Que problemas podem ocorrer com laços múltiplos?

Definir uma matriz permite exercitar seus conhecimentos sobre os comandos **FOR...NEXT** e **IF...THEN** (tente contar quantos desses comandos existem nos programas apresentados aqui; você logo vai perceber que eles são muitos). Porém, a parte mais difícil da programação de matrizes diz respeito ao uso de laços **FOR...NEXT** múltiplos, ou "aninhados" (isto é, um dentro do outro). Com efeito, é nessa parte que os iniciantes, e mesmo os programadores experientes, mais se confundem.

Examine as linhas dos programas do artigo. Elas são responsáveis pela leitura dos dados armazenados nas linhas **DATA**.

Cada linha de **DATA** corresponde a uma linha da matriz de dados, de modo que o **FOR...NEXT** que controla a leitura das linhas deve vir antes (laço mais externo). Se a linha de **DATA** contivesse os elementos de uma coluna da matriz, teríamos que usar como **FOR...NEXT** externo aquele que lê as colunas.

É por isso que o conjunto é dimensionado como **N (L,C)**, em vez de **N (C, L)**. A ordem dos índices de linha e coluna no comando **DIM** não é muito importante, desde que você os faça corresponder aos laços **FOR...NEXT**.

Os cabeçalhos para as linhas e colunas devem ser definidos separadamente, para facilitar a compreensão.

```
850 IF fo=0 THEN  PRINT ''"NOM
E NAO ENCONTRADO.TENTE OUTRA V
EZ": GOTO 800
860 FOR j=1 TO c
870 IF n(fo,j)>0 THEN  PRINT n
(fo,j);" ";p$(j)
875 NEXT j
880 PRINT ''" PRESSIONE QUALQU
ER TECLA PARA   RETORNAR"
885 PAUSE 0
890 RETURN
```

```
799 REM ### OPCAO 4 ###
800 PRINTTAB(2);"DIGITE O NOME
DE UMA CRIANÇA"
805 PRINT"LISTA DE TODOS OS ANI
MAIS QUE PERTENCEM A ";
810 INPUT F$
815 PRINT:PRINT
820 FORJ=1TOR
830 IFCH$(J)=F$THENFO=J
840 NEXT
850 IF FO=0THENPRINT:PRINT"NOME
 NÃO ACHADO. TENTE NOVAMENTE!":
GOTO800
860 FORJ=1TOC
870 IFN(FO,J)>0THENPRINTN(FO,J)
;" ";PET$(J)
875 NEXT
880 PRINT:PRINT"PRESSIONE QUALQ
UER TECLA PARA RETORNAR";
885 IFINKEY$=""THEN885
890 RETURN
```

```
799   REM   *** OPCAO 4 ***
800   PRINT  TAB( 5);"DIGITE O N
OME DE UMA CRIANCA"
805   PRINT "LISTA DE TODOS OS A
NIMAIS QUE PERTENCEM A ";
810   INPUT F$
815   PRINT : PRINT
820   FOR J = 1 TO R
830   IF CH$(J) = F$ THEN FO = J

840   NEXT
850   IF FO = 0 THEN  PRINT : PR
INT "NOME NAO ENCONTRADO. TENTE
 NOVAMENTE!": GOTO 800
860   FOR J = 1 TO C
870   IF N(FO,J) > 0 THEN  PRINT
 N(FO,J);" ";PET$(J)
875   NEXT
880   PRINT : PRINT "PRESSIONE Q
UALQUER TECLA PARA RETORNAR";
885   GET A$
890   RETURN
```

A opção 5 pede que você entre um número e imprime a lista de pessoas que têm no mínimo esse número de animais. Ela também informa quantos animais cada pessoa tem.

Digite então a próxima sub-rotina da seguinte forma:

```
899 REM ** OPCAO 5 **
900 PRINT '" DIGITE UM NUMERO
PARA LISTAR    TODOS OS QUE TE
M AO MENOS ESTE  NUMERO DE ANI
MAIS"
910 INPUT a
915 PRINT ''
920 FOR j=1 TO r
930 LET p(j)=p(j)+n(j,k)
935 NEXT k
940 LET p(j)=0
945 LET p(j)=0
950 NEXT j
955 IF fo=0 THEN  PRINT ''"NIN
GUEM TEM TANTOS ANIMAIS !"
960 PRINT ''" PRESSIONE QUALQU
ER TECLA PARA   RETORNAR"
970 PAUSE 0
980 RETURN
```

```
899 REM ### OPCAO 5 ###
900 PRINT"DIGITE UM NÚMERO PARA
 LISTAR TODOS QUE TÊM AO MENOS
ESTE NÚMERO DE ANIMAIS";
910 INPUTA
915 PRINT:PRINT
920 FORJ=1TOR
925 FORK=1TOC
930 PT(J)=PT(J)+N(J,K)
935 NEXTK
940 IFPT(J)>=ATHENPRINTCH$(J);"
 ";PT(J):FO=1
945 PT(J)=0
950 NEXTJ
955 IFFO=0THENPRINT:PRINT"NINGU
ÉM TEM TANTOS ANIMAIS!"
960 PRINT:PRINT"PRESSIONE QUALQ
UER TECLA PARA RETORNAR"
970 IF INKEY$=""THEN970
980 RETURN
```

```
899   REM   *** OPCAO 5 ***
900   PRINT "DIGITE UM NUMERO PA
RA LISTAR TODOS QUE TEM AO MENO
S ESTE NUMERO DE ANIMAIS";
910   INPUT A
915   PRINT : PRINT
920   FOR J = 1 TO R
925   FOR K = 1 TO C
930 PT(J) = PT(J) + N(J,K)
935   NEXT K
940   IF A <  = PT(J) THEN  PRIN
T CH$(J);" ";PT(J):FO = 1
945 PT(J) = 0
950   NEXT
955   IF FO = 0 THEN  PRINT : PR
INT "NINGUEM TEM TANTOS ANIMAIS
!"
960   PRINT : PRINT "PRESSIONE Q
UALQUER TECLA PARA RETORNAR";
970   GET A$
980   RETURN
```

Antes de comparar o número que você imprimiu com os totais para cada criança, o computador precisa calcular esses totais. As linhas 920 e 925 definem os laços que percorrem as linhas e colunas da matriz, enquanto a linha 930 cal-

cula o total de animais em cada linha. Os totais são armazenados em um conjunto chamado **PT( )**, ou **p( )**, no Spectrum. Se o total de cada linha for maior ou igual ao número que você entrou, o nome da criança será exibido na tela, juntamente com o número de animais que ela possui.

Assim que o nome de alguém for impresso, o indicador **FOUND** (ou seja, achado, em inglês) será igualado a 1. Se **FOUND** ainda for igual a 0 quando o computador atingir a linha 955, isso significa que nenhuma criança tem aquele número de animais que você solicitou, e o programa lhe dará essa informação por meio de mensagem.

A última opção — de número 6 — mostra a matriz na tela.

Como é difícil encaixar todos os nomes dos animais de estimação no topo da tabela, o programa imprime números de referência, em vez de palavras. Em compensação, ele imprime os nomes dos proprietários.

```
 999 REM ** OPCAO 6 **
1000 PRINT '"CRIANCAS","ANIMAIS
"
1010 PRINT 'TAB 9;
1015 FOR j=1 TO c
1020 PRINT j;"  ";
1025 NEXT j: PRINT
1030 FOR j=1 TO r
1035 PRINT 'c$(j);TAB 9;
1040 FOR k=1 TO c
1050 PRINT n(j,k);"  ";
1060 NEXT k
1065 NEXT j
1070 PRINT ''" PRESSIONE QUALQU
ER TECLA PARA   RETORNAR"
1080 PAUSE 0
1090 RETURN
```

```
999 REM### OPCAO 6 ###
1000 PRINT @32,"CRIANCAS","ANIM
AIS"
1010 PRINT @72,;
1015 FOR J=1 TO C
1020 PRINT J
1025 NEXT:PRINT
1030 FOR J=1 TO R
1035 PRINT:PRINT CH$(J) TAB(8);
1040 FOR K=1 TO C
1050 PRINT N(J,K);
1060 NEXT K,J
1070 PRINT:PRINT:PRINT " PRESSI
ONE QUALQUER TECLA PARA   RETOR
NAR"
1080 IF INKEY$="" THEN 1080
1090 RETURN
```

```
999  REM  *** OPCAO 6 ***
1000  PRINT "CRIANCAS","ANIMAIS
"
1010  HTAB 15
1015  FOR J = 1 TO C
1020  PRINT J;" ";
1025  NEXT : PRINT
1030  FOR J = 1 TO R
1035  PRINT : PRINT CH$(J); TAB
( 15);
1040  FOR K = 1 TO C
1050  PRINT N(J,K);" ";
1060  NEXT : NEXT
1070  PRINT : PRINT : PRINT "PR
ESSIONE QUALQUER TECLA PARA RET
ORNAR";
1080  GET A$
1090  RETURN
```

```
999 REM ### OPCAO 6 ###
1000 PRINT"CRIANÇAS","ANIMAIS"
1010 LOCATE 15
1015 FORJ=1TOC
1020 PRINTJ;
1025 NEXT:PRINT
1030 FORJ=1TOR
1035 PRINT:PRINTCH$(J);TAB(15);
1040 FORK=1TOC
1050 PRINTN(J,K);
1060 NEXT:NEXT
1070 PRINT:PRINT:PRINT"PRESSION
E QUALQUER TECLA PARA RETORNAR"
;
1080 IFINKEY$=""THEN1080
1090 RETURN
```

USOS PARA AS MATRIZES

Uma matriz usada dessa forma é chamada de base de dados. É possível criar bases de dados para os mais diversos fins: para um levantamento da vida animal, por exemplo. Numa pesquisa desse gênero, a base de dados mostraria o número de certos animais, assim como o tipo de habitat onde eles são encontrados. As linhas da matriz indicariam os tipos de animais, e as colunas, os habitats. Em seguida, os dados poderiam ser examinados para verificar a distribuição dos animais em determinadas regiões ou épocas do ano.

A base de dados pode ser aplicada também em levantamentos de tráfego em estradas: para verificar, por exemplo, que tipos de carros passam por cada seção de uma rodovia. Ela se presta ainda para aplicação no campo da nutrição. Assim, a matriz conteria os dados de cada tipo de alimento, com relação ao seu conteúdo de proteínas, gorduras, carboidratos e calorias. Com uma matriz desse tipo, seria fácil imprimir todos os alimentos que apresentassem uma certa quantidade de proteínas, ou tantas calorias, ou que oferecessem certas combinações de elementos.

COMO USAR A DECLARAÇÃO *REM*

Não basta que o programa em BASIC esteja correto ou passe por todos os testes: é preciso ainda que ele esteja *bem documentado* e estruturado de forma lógica e clara.

Para evitar confusões por parte do usuário e mesmo do programador na compreensão do que foi digitado, costuma-se utilizar um mecanismo conhecido como *documentação interna* do programa. Este identifica, por meio de títulos e comentários colocados na listagem, as diversas partes do programa. Isso é feito pelo comando **REM**, que é uma abreviatura da palavra *REMark* (*comentário*, em inglês). O **REM** não é uma declaração executável (ou seja, ao encontrá-la em um programa, o computador simplesmente a ignora e passa adiante).

Uma das funções do **REM** consiste em identificar o programa, prestando informações (chamadas *cabeçalho*) como: para que serve, para que máquina ele está destinado, quem o escreveu, quando etc. Eis um exemplo:

```
10 REM ----------------------
15 REM PROGRAMA CREPE
25 REM Simular o jogo de crepe
30 REM VERSAO:1DATA:20/02/1986
35 REM NOME P/ GRAVACAO: CREP01
40 REM
45 REM COMPUTADOR: TK-90X
50 REM MEMORIA NECESSARIA:16 K
55 REM PROGRAMADO POR: RENATO
       SABBATINI
60 REM ----------------------
```

A colocação de cabeçalhos como esse evita muitos dissabores.

Outra função do **REM** é informar para que servem certas partes do programa. Por exemplo:

```
350 REM --------------------
355 REM TESTA SE NUMERO DE NA
        VES=ZERO
360 REM SE E', TERMINA O PRO-
        GRAMA
365 REM --------------------
370 IF NAVES=0 THEN GOTO 999
```

O ideal seria colocar um comentário para cada linha do programa. Isso pode ser feito facilmente nos micros que aceitem comandos múltiplos por linha:

```
500 GOSUB 890 :REM SUBR.
CALCULO
```

O emprego de comentários é simplificado, em vários micros, pelo apóstrofo ('), usado como abreviatura do comando **REM**:

```
10 ' Isto é um comentário
```

COMO PLANEJAR UMA AVENTURA

Sair da rotina e fazer da vida uma aventura errante: quantos de nós já não sonhamos com isso? Veja como fazer essa mudança, vivendo aventuras incríveis... no computador.

Os jogos de aventura são uma boa alternativa para quem está cansado dos arcaicos videogames e fliperamas. Neles, o jogador é totalmente envolvido por um mundo fantástico, criado unicamente pela imaginação do programador.

A ORIGEM DOS JOGOS DE AVENTURAS

A idéia de criar jogos de aventura surgiu da popularidade, nos EUA e na Grã-Bretanha, dos jogos de ação não-computadorizados, tais como *Masmorras e Dragões* e, ao mesmo tempo, do desejo de se utilizar os computadores para fazer coisas mais interessantes do que processar dados.

Por outro lado, os jogos não-computadorizados não atendiam plenamente à demanda do público, que exigia emoções cada vez mais fortes. Essa demanda só foi satisfeita pelos jogos de aventuras programados para o computador.

Em *Masmorras e Dragões* os jogadores assumem o papel de certos personagens, que lutam encarniçadamente (na imaginação) num lugar conhecido como *Masmorra*. Esse lugar, por sua vez, é criado por outro jogador, que faz o papel do *Senhor da Masmorra*. Nos jogos de aventura, o programador vive um personagem parecido com este, podendo inventar, assim, seu próprio mundo.

Ao contrário de outros jogos, em *Masmorras e Dragões* os jogadores não podem escolher as características de seus personagens; isso depende inteiramente do jogo. Em algumas versões mais sofisticadas, os jogadores podem selecionar o equipamento e os mantimentos que os personagens vão usar, desde que isso seja feito no início do jogo, antes de começar a aventura. O primeiro jogo de aventura destinado a computadores foi escrito para máquinas de grande porte, em linguagem FORTRAN, e não em BASIC. O programa ocupou aproximadamente 300k de memória, ou seja, bem mais do que a memória RAM total da maioria dos micros pessoais. Apesar do seu tamanho e complexidade, esse jogo conquistou milhares de usuários de computadores nos Estados Unidos e no Canadá; eles passaram a dedicar suas noites a tentar decifrar os mistérios do jogo.

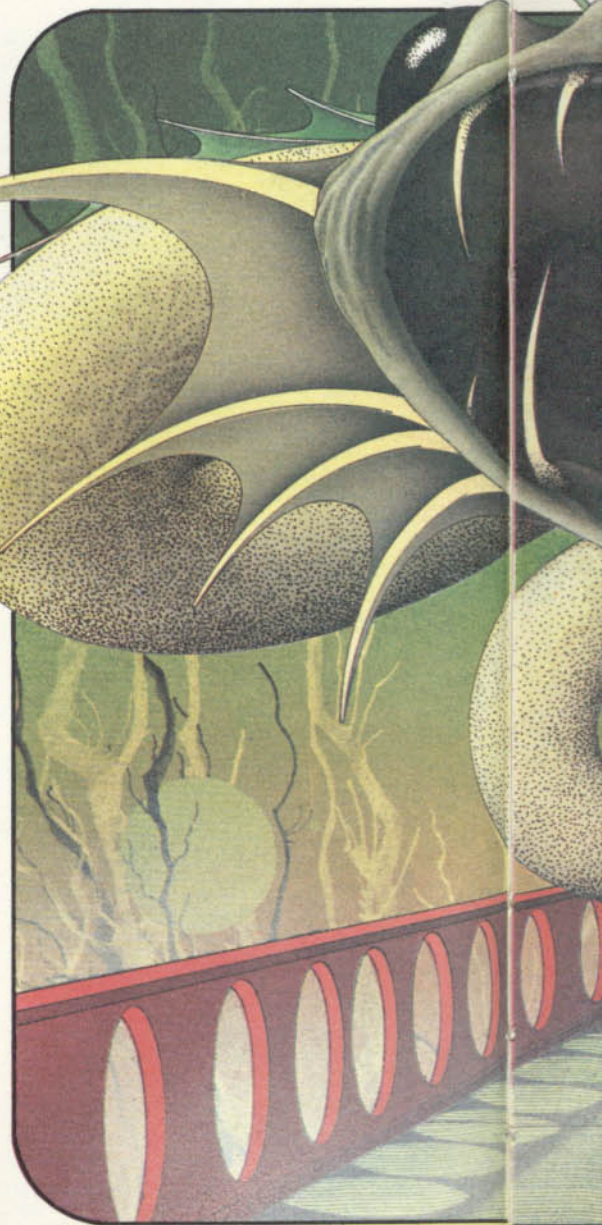
Na realidade, a popularização dos jogos de aventura começou quando um jovem programador chamado Scott Adams desenvolveu, em 1978, um jogo para microcomputador TRS-80 e provou que era possível escrever uma aventura razoável em um espaço de memória bem menor. Desde então, os temas de jogos adotados por Adams — A Terra da Aventura, O Refúgio do Pirata, O Mistério do Parque de Diversões etc. — têm sido usados e recusados inúmeras vezes pelos escritores de outros jogos.

TIPOS DE AVENTURAS

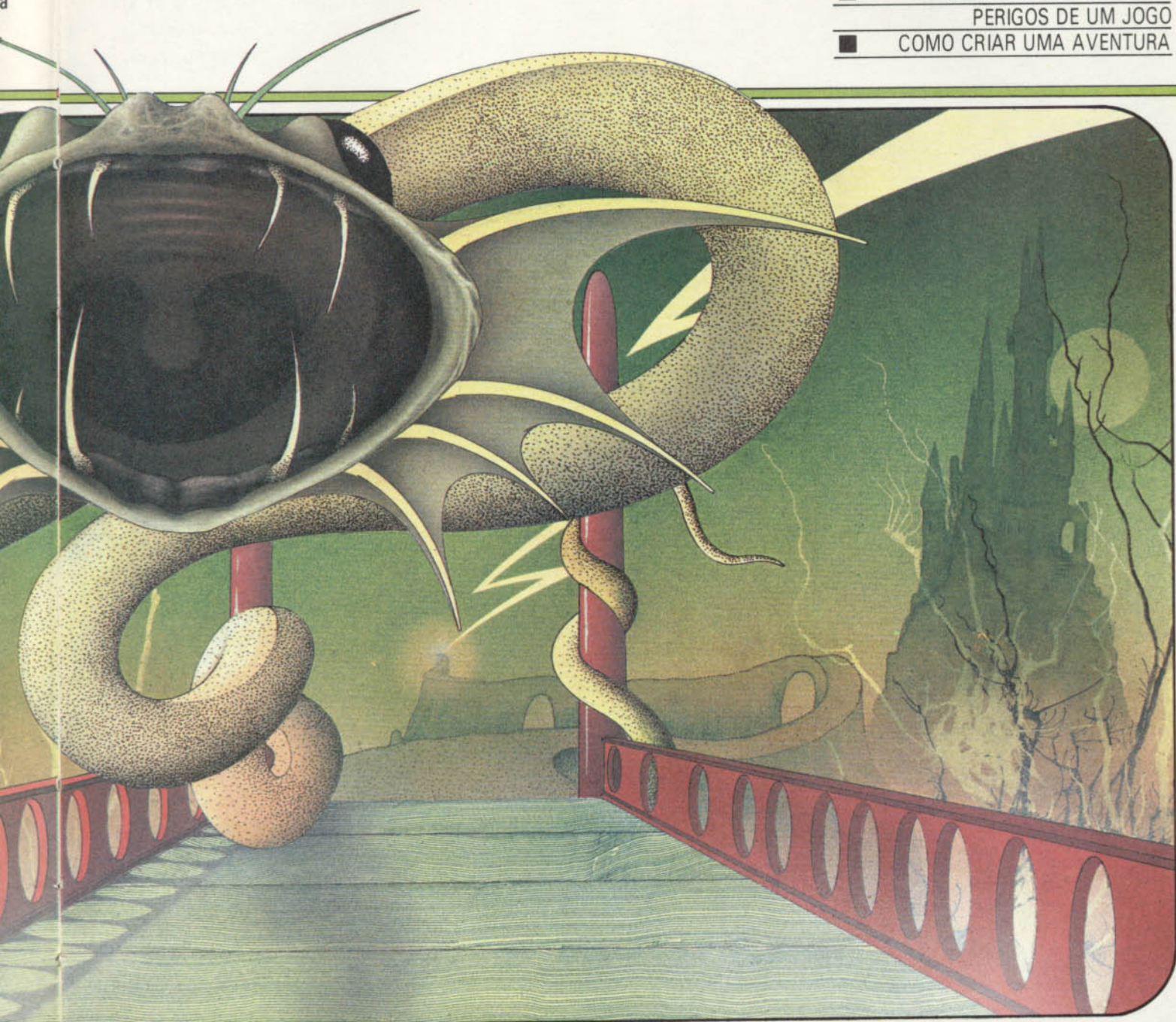
Como as aventuras para grandes computadores, os jogos de Adams apresentavam somente textos na tela. Na verdade, os programas que usam apenas textos são ainda os mais populares.

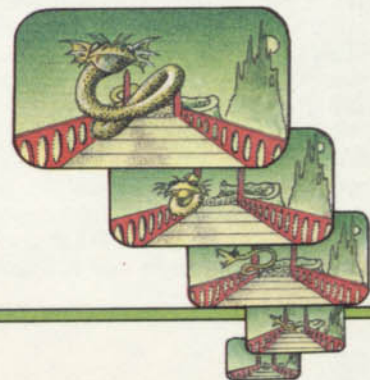
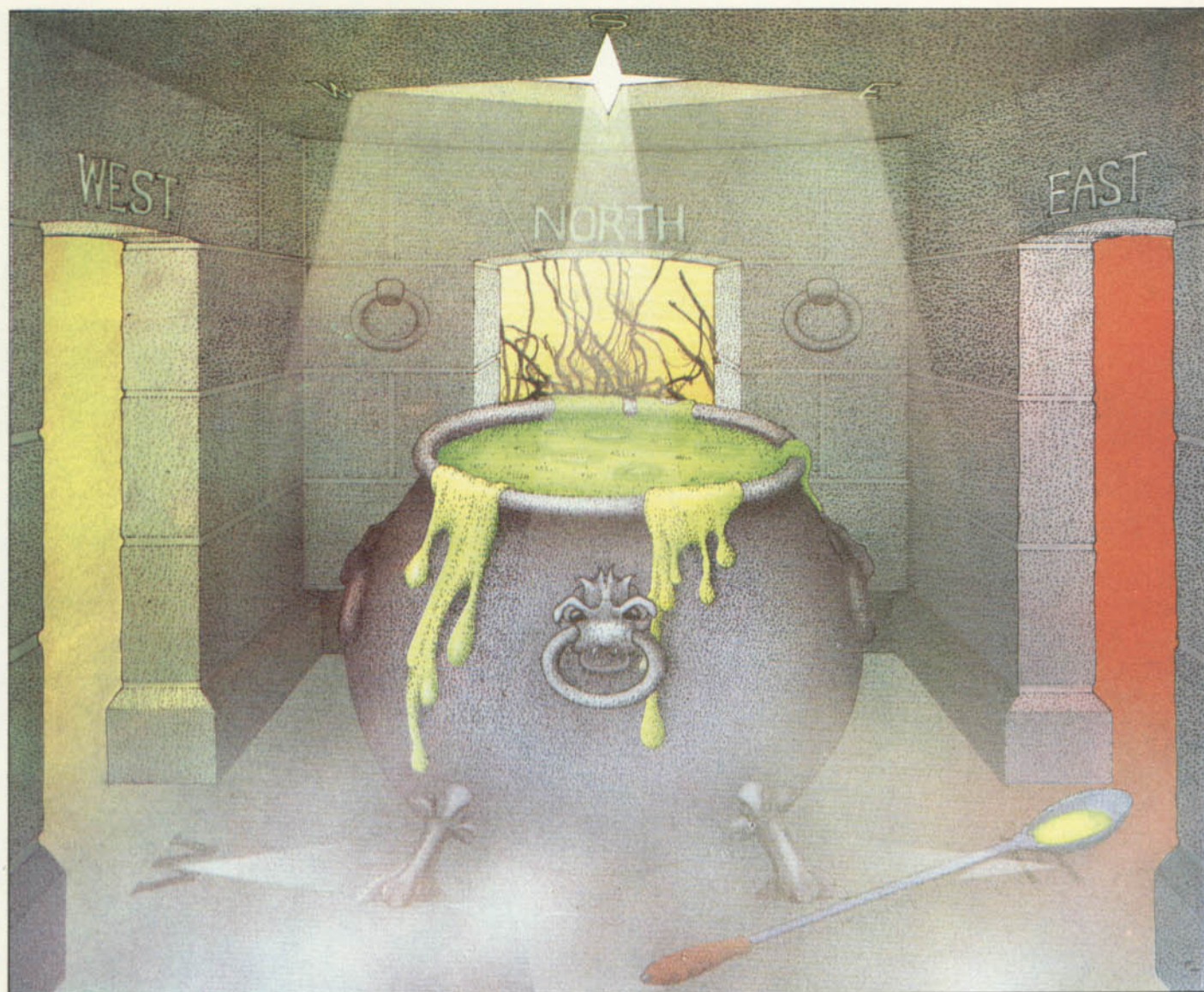
Existem ainda inúmeros jogos de aventura para computadores que utilizam ilustrações desenhadas na tela para situar, orientar e motivar o usuário. Entretanto, os gráficos precisam ser muito sofisticados, para poder competir com a imaginação do jogador. Por exemplo, você poderia conceber um monstro bem mais horripilante do que seria capaz de reproduzir, mesmo nas telas gráficas mais avançadas. Portanto, é bem possível que os gráficos "estraguem" sua diversão: o principal argumento contra a sua utilização é que eles desperdiçam grande parte da memória, que poderia ser usada para expandir o campo da aventura. Certas aventuras apresentam uma contagem de pontos, sempre que uma etapa da busca é completada. Desse modo, o jogador pode avaliar seu desempenho, caso tenha sido derrotado em algum estágio. Alguns programas chegam ao ponto de classificar esse desempenho com conceitos, que podem ir de "palhaço" a "perito".

Outros jogos não fornecem qualquer dica de como o jogador está se saindo ou da proximidade do desfecho.

Seja qual for o caso, porém, a diversão maior está na solução de uma série aparentemente infinita de charadas, depois da qual a aventura chega ao fim. Alguns costumam durar muitas horas, ou dias, podendo ser interrompidos e retomados a qualquer momento.

UMA HISTÓRIA SEM FIM

Normalmente, quando se inicia uma aventura através do comando **RUN**, são

- JOGOS DE AVENTURA
- COMO JOGAR
- DICAS PARA VENCER OS PERIGOS DE UM JOGO
- COMO CRIAR UMA AVENTURA

dadas informações sobre o mundo em que se acaba de entrar: este pode ser um lugar exótico na Terra, um planeta distante, ou um local imaginário. O jogo pode acontecer em épocas diversas. Geralmente, são fornecidas informações como: quem governa o lugar; quem é você e como fazer para atingir o objetivo e ganhar o jogo. Depois disso, aparecerá uma primeira descrição do local, provavelmente dizendo algo como:

```
 VOCÊ SE ENCONTRA PRÓXIMO A UM
ENORME CALDEIRÃO CHEIO DE UM
LÍQUIDO VERDE BORBULHANTE. HÁ
UM CHEIRO DEMONÍACO NO AR. UMA
COLHER ESTÁ NO CHÃO.
VOCÊ PODE IR PARA O LESTE
                    OESTE
                    NORTE
E AGORA?
```

Agora, você tem que decidir o que fazer. Será conveniente usar a colher para misturar o líquido ou mesmo experimentá-lo? Não seria melhor deixar isso de lado? Ou talvez fosse preferível procurar um recipiente para levar um pouco do líquido verde com você?

Se decidir usar a colher, digite algo assim: **PEGUE COLHER**; a resposta do computador será: OK ou VOCÊ AINDA NÃO PODE PEGAR A COLHER!, ou alguma outra mensagem.

Em cada etapa do jogo você deve informar ao computador exatamente o que pretende fazer. A forma de fazer isso dependerá apenas do tipo de jogo. A maioria deles espera você digitar seus comandos no computador através de um verbo seguido de substantivo; por exemplo: **PEGUE COLHER**.

Jogos mais sofisticados aceitam sentenças completas, mas isso é uma exceção à regra. Esses tipos de jogos permitem-lhe dizer algo como: **MATE O INSETO PISANDO NELE ENQUANTO CANTA "O TICO-TICO NO FUBÁ"**.

A maioria dos jogos de aventura consegue entender abreviaturas das palavras válidas. Por exemplo, é muito comum, numa aventura, digitar **N** em vez de **NORTE**. As direções podem ser pontos cardeais — **N, S, L** e **O** — ou colaterais — **SE, SO, NE** e **NO** —, ou instruções como **PARA CIMA, PARA BAIXO**.

Um jogo típico de aventura é baseado numa grade de locais possíveis, que assume geralmente a forma de um quadrado. Dependendo da imaginação do programador, esses lugares podem representar ambientes tão variados como os quartos de um castelo, os subterrâneos de uma mina etc.

DICAS PARA VENCER

Habitualmente, existe apenas uma solução para a aventura: por exemplo, encontrar um pote de ouro e levá-lo para o fim do arco-íris, ou matar um guerreiro medieval e escapar ileso etc. Até chegar a esse objetivo, o jogador precisa resolver toda uma série de problemas. O mais provável é que ele tenha que fazer muitas tentativas diferentes até concluir a aventura.

Existem algumas regras básicas e dicas que o ajudarão a resolver mais rapidamente a maioria dos jogos. Quase todos os objetos que você encontrar nas aventuras serão de alguma utilidade. A existência de muitas pistas completamente falsas representa um desperdício de memória, embora seja necessário es-

tar sempre atento para alguns objetos que podem ser "facas de dois gumes". Por exemplo, o jogador poderia estar carregando uma sacola com moedas de ouro para passar pelo pedágio de uma ponte; caso ele se decidisse a nadar no rio, porém, o peso delas o faria afundar. Como regra geral, é aconselhável tentar carregar o maior número possível de objetos, mas algumas vezes apenas uma parte deles lhe será realmente útil.

Quase sempre, a maioria dos objetos é usada somente uma vez durante uma aventura. Mas há exceções: uma espada, por exemplo, poderia ser usada muitas vezes para derrubar gigantes, dragões ou bandidos. Dessa forma, se você for obrigado a limitar o número de objetos que leva consigo, deve ter em mente que é mais seguro descartar os que já foram usados uma vez.

Outra regra geral: desenhe *sempre* um mapa, marcando nele os nomes dos aposentos, a posição dos sentinelas e dos objetos distribuídos pelos quartos, *todas* as entradas e saídas com suas direções e outros pontos de referência que lhe possam ser úteis.

O mapa o fará economizar tempo e esforço quando você estiver retornando para algum ponto anterior — manobra que você repetirá várias vezes durante o jogo. Se tiver que abandonar algum objeto, por não poder carregar tudo, não se esqueça de marcar a posição dele no mapa. A importância do mapa está em que ele lhe permitirá ter certeza de que explorou todas as possibilidades da aventura — o que, em muitos casos, salvará sua "vida".

Certos jogos permitem que você peça um inventário do que está carregando. Assim, quando estiver diante de uma charada, procure saber exatamente o que tem em mãos, digitando **INVENTÁRIO**, **LISTA**, ou algo semelhante, dependendo da aventura.

Há jogos, ainda, que dão ao jogador a possibilidade de pedir ajuda. A forma de fazê-lo varia, mais uma vez, conforme o conteúdo da aventura. Freqüentemente, porém, você obterá algo assim, em resposta ao seu pedido:
NENHUMA AJUDA POSSÍVEL.

Alguns jogos seguem cuidadosamente o enredo de um livro específico (por exemplo, *A Ilha do Tesouro*, escrito em 1833 por Robert Louis Stevenson). Nestes casos, é altamente recomendável ler o livro em questão.

Outros jogos inspiram-se apenas em alguns capítulos ou idéias de um romance famoso. Assim, se você encontrar numa aventura algum elemento que o faça recordar um texto literário e tiver dificuldades para resolver um determinado problema, tente encontrar a resposta consultando o livro. Um dicionário de sinônimos pode ser de grande utilidade, permitindo que o jogador explore todas as variações possíveis de uma determinada frase. Por exemplo: o programador pode ter usado no jogo a palavra POLIR, em vez de LUSTRAR.

Uma última dica: se o jogo permitir o uso do comando **GRAVAR** em algumas etapas, e você estiver prestes a enfrentar um grande perigo, então armazene a aventura no estágio em que ela se encontra, antes de dar o passo decisivo, pois, se você for "morto", poderá voltar atrás e continuar a partir do ponto em que parou.

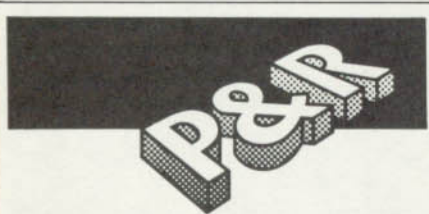
CRIE AS SUAS PRÓPRIAS AVENTURAS

A criação de aventuras é uma boa maneira de dominar a linguagem BASIC. Quase todos os aspectos importan-

tes dessa linguagem são utilizados nesse tipo de jogo: variáveis, matrizes, formatos de telas, cadeias alfanuméricas, comandos **IF**, e assim por diante.

Muitos jogos de aventuras à venda ainda são escritos em BASIC, pois não há muita necessidade de velocidade de execução do programa.

Antes de programar um jogo desse gênero, você precisa ter uma idéia bem clara do que pretende fazer e do enredo da aventura. A seguir, planeje cuidadosamente a trama com suas charadas, perigos, objetos etc.

Para começar, pegue uma folha de papel e faça um rascunho de algumas idéias. Não se preocupe se ainda não tem uma visão completa da ordem do jogo: o que você precisa inicialmente é de uma noção do roteiro, do local da aventura e de algumas charadas para o jogador resolver.

Quanto espaço de memória necessito para escrever um jogo de aventura?

Nas próximas lições de INPUT desenvolveremos uma aventura simples, ocupando cerca de 5K de BASIC. Como todo primeiro passo, esse jogo não trará episódios de grande emoção. Seu objetivo é apenas introduzi-lo ao mundo fascinante da aventura.

Os melhores jogos têm um grande número de aposentos e charadas e são muito difíceis, devido ao efeito crescente de problemas cada vez mais complicados. Com uma quantidade limitada de memória, você será forçado a determinar um número menor de problemas, que por sua vez serão mais difíceis do que o normal. Isso, contudo, não é muito satisfatório.

Um jogo como esse exige um computador com, no mínimo, 16K de memória; mas sugerimos uma máquina com 48 ou 64K para jogos realmente desafiantes.

O que devo fazer se cair numa armadilha durante a aventura?

Não desista; tente novamente quando tiver uma nova idéia ou estratégia.

Se você realmente empacar, os jogos de aventuras comercializados oferecem um folheto com dicas e respostas. Muitas vezes, é preciso solicitar pelo correio um mapa da solução.

Outro recurso para ajudá-lo são as revistas do gênero, que costumam apontar a solução completa de aventuras complexas.

Além de livros, você pode inspirar-se em filmes, programas de TV e peças teatrais, ou mesmo inventar novas histórias. Pode também buscar idéias em outras aventuras. Mas o melhor recurso para sua inspiração é fazer trabalhar a imaginação.

Ao mesmo tempo, procure alcançar um certo equilíbrio entre o desafio e as condições para vencê-lo. Não é conveniente criar uma aventura que qualquer um poderia resolver no espaço de meia hora. Em contrapartida, seu jogo não deve apresentar problemas insolúveis. A regra básica é: "dê aos jogadores uma chance"... mas não exagere.

Tente não deixar muitos aposentos ou locais vazios. Além de ocupar muito espaço de memória, estes não contribuem para a resolução dos problemas e tornam o enredo cansativo.

Por outro lado, procure evitar que as suas primeiras aventuras sejam excessivamente complexas, para que os problemas que aparecem ao longo da história possam ser resolvidos por quem ainda não tem prática; familiarize-se com o tema antes de aventurar-se em qualquer enredo extravagante; mantenha-se alerta quanto ao espaço restante de memória, pois programas como esses tendem a se tornar gigantescos.

No jogo de aventura que será elaborado nas próximas partes do curso de *Programação de jogos*, encontraremos muitas declarações **REM** (comentários, para entender o fluxo do programa). Entretanto, para economizar memória em jogos com longas aventuras, é preferível trabalhar sem tais declarações, mesmo que no começo elas facilitem a criação da trama.

Você poderá também poupar espaço de memória na descrição dos aposentos. Descrições muito curtas, porém, tendem a estragar todo o prazer suscitado pela aventura.

O ESPAÇO RESTANTE DE MEMÓRIA

Ao escrever um longo jogo de aventura, é fácil descobrir quando se está chegando ao limite de memória do computador. Obviamente, os problemas de desperdício de memória tornam-se mais graves no caso de máquinas com memórias menores (na verdade, seria perda de tempo tentar escrever um programa de aventura em um computador que tenha menos do que 16K de RAM).

Digite a seguinte linha, em modo direto:

```
PRINT (PEEK 23730 + 256* PEEK
23731) - (PEEK 23653 + 256*
PEEK 23654)
```

O que foi digitado leva em consideração tanto o espaço ocupado pelo programa como o usado para armazenar as variáveis. A melhor contagem para descobrir a quantidade de memória restante só será possível, portanto, depois que o programa tiver sido rodado.

Para saber quanto espaço está sobrando, subtraia esse valor da quantidade total de memória RAM existente no computador.

Nos microcomputadores das linhas TRS-80 e TRS-Color, o que resta da memória pode ser descoberto digitando-se:

```
PRINT MEM
```

Essa instrução levará em conta o espaço ocupado pelo programa e pelas variáveis. Assim, é melhor rodar antes o programa, durante seu desenvolvimento (se isso for possível), a fim de conseguir uma indicação real do total de memória usado por ele.

Para saber quanto espaço está sobrando, subtraia esse valor da quantidade total de memória RAM existente no computador. Há uma maneira bem fácil de se aumentar o espaço livre de memória no TRS-Color para seus programas em BASIC. Antes de começar a programar, digite esses dois comandos **POKE**:

```
POKE 25,6
POKE 26,1
NEW
```

Isso economizará uns 6K extras para seu programa, pois enganará o computador, colocando o programa em BASIC dentro de um espaço normalmente reservado para gráficos de alta resolução.

Neste caso os comandos **POKE** excluem os comandos para gráficos na sua aventura. Se eles existirem, o seu programa será alterado.

Quando você armazenar o programa em fita ou disco e quiser carregá-lo mais tarde, não se esqueça de usar antes os comandos **POKE** e **NEW**.

Nos microcomputadores das linhas Apple e MSX, o que resta da memória pode ser descoberto digitando-se:

```
PRINT FRE(0)
```

A função **FRE** (abreviatura de **FREe**, ou livre, em inglês) retorna o espaço livre na memória RAM, ou seja, aquele não ocupado pelo programa e pelas variáveis.

PROGRAMAS EM CÓDIGO DE MÁQUINA

- EXERCÍCIO EM ASSEMBLY
- UM PROGRAMA QUE DESLOCA A TELA PARA OS LADOS
- COMO FUNCIONA O PROGRAMA EM DIVERSOS COMPUTADORES

Agora que você já sabe traduzir do Assembly para o sistema hexa, passe a operar com código de máquina.

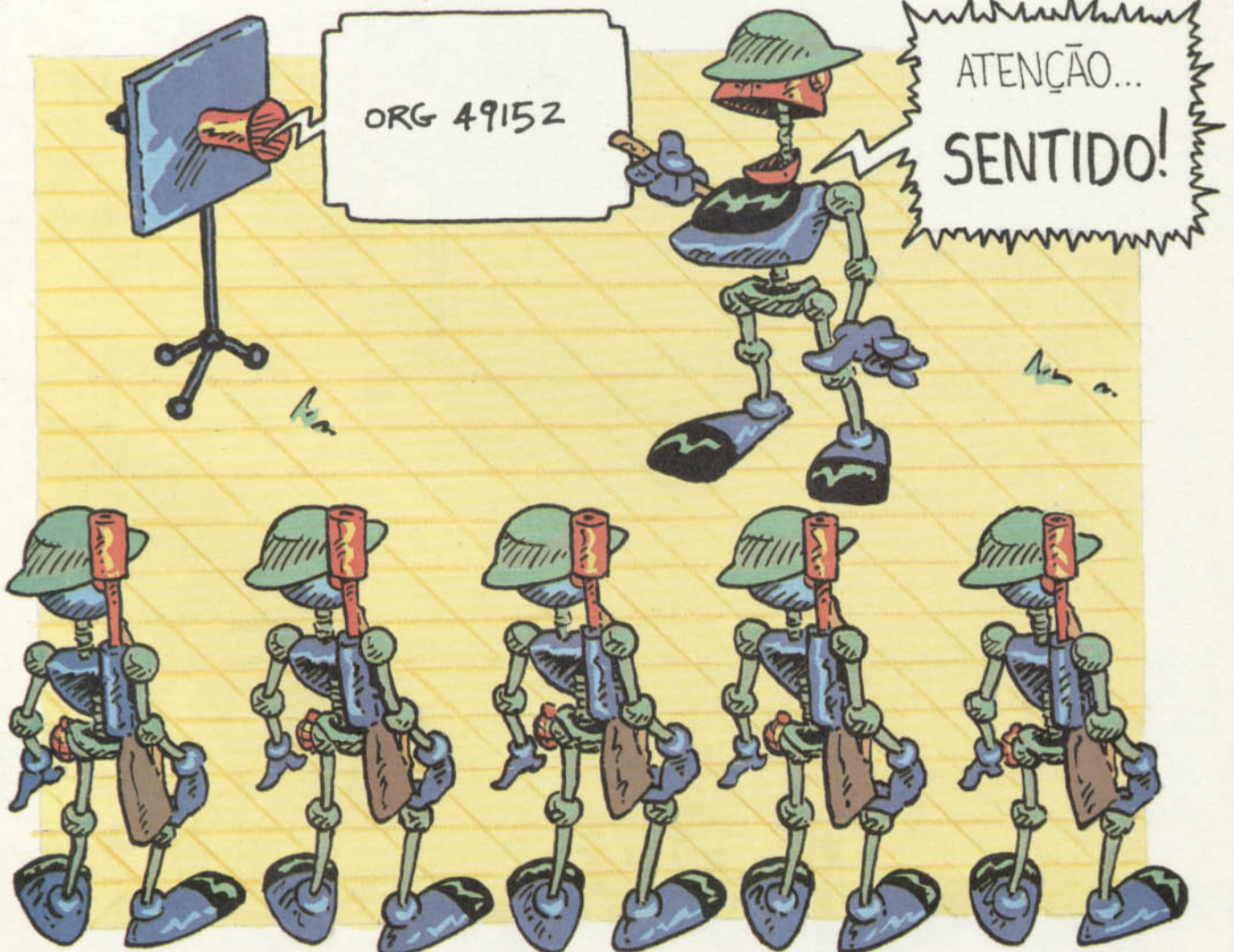
Os programas em Assembly apresentados a seguir deslocam a tela um caractere a esquerda ou para a direita. Monte-os na memória do micro usando seu programa monitor. Se você quiser gravar programas para uso futuro, coloque-os em locais diferentes da memória.

Os programas funcionam independentemente de suas posições na memória, de forma que você mesmo pode resolver onde é melhor colocá-los. Para deslocar a tela mais de uma posição no sentido horizontal, chame a rotina em código de máquina de dentro de um laço **FOR...NEXT**. Outra opção é deslocar a tela somente quando certa tecla é pressionada, usando **INKEY$** ou **GET$**.

S

Este programa desloca a tela para a esquerda (já está traduzido).

```
     ld de,16384      11 00 40
     ld hl,16385      21 01 40
     ld b,192            06 C0
loop push bc                C5
     ld a,(de)              1A
     ld bc,31         01 1F 00
     ldir                ED B0
     ld(de),a               12
     inc de                 13
     inc hl                 23
     pop bc                 C1
     djnz loop           10 F3
     ret                    C9
```

Os comandos **ld de, 16384** e **ld hl, l, 16385** carregam os endereços das duas

primeiras posições da tela do Spectrum nos registros DE e HL. Já **ld b,192** transporta o registro B com o número 192 (há 192 linhas na tela desse micro, e B é usado para contá-las).

Como o registro B será usado para contar outras coisas também, seu conteúdo ficará temporariamente armazenado na pilha (*stack*). Todavia, nenhum comando transfere apenas o conteúdo de B para a pilha: assim, temos que fazer essa transferência com o conteúdo de B e C ao mesmo tempo usando **push bc**.

O acumulador é então carregado com o conteúdo da posição de memória cujo endereço está no registro DE, pelo comando **ld a (DE)**. Este é um exemplo de endereçamento indireto. Lembre-se de que o conteúdo de DE é 16384.

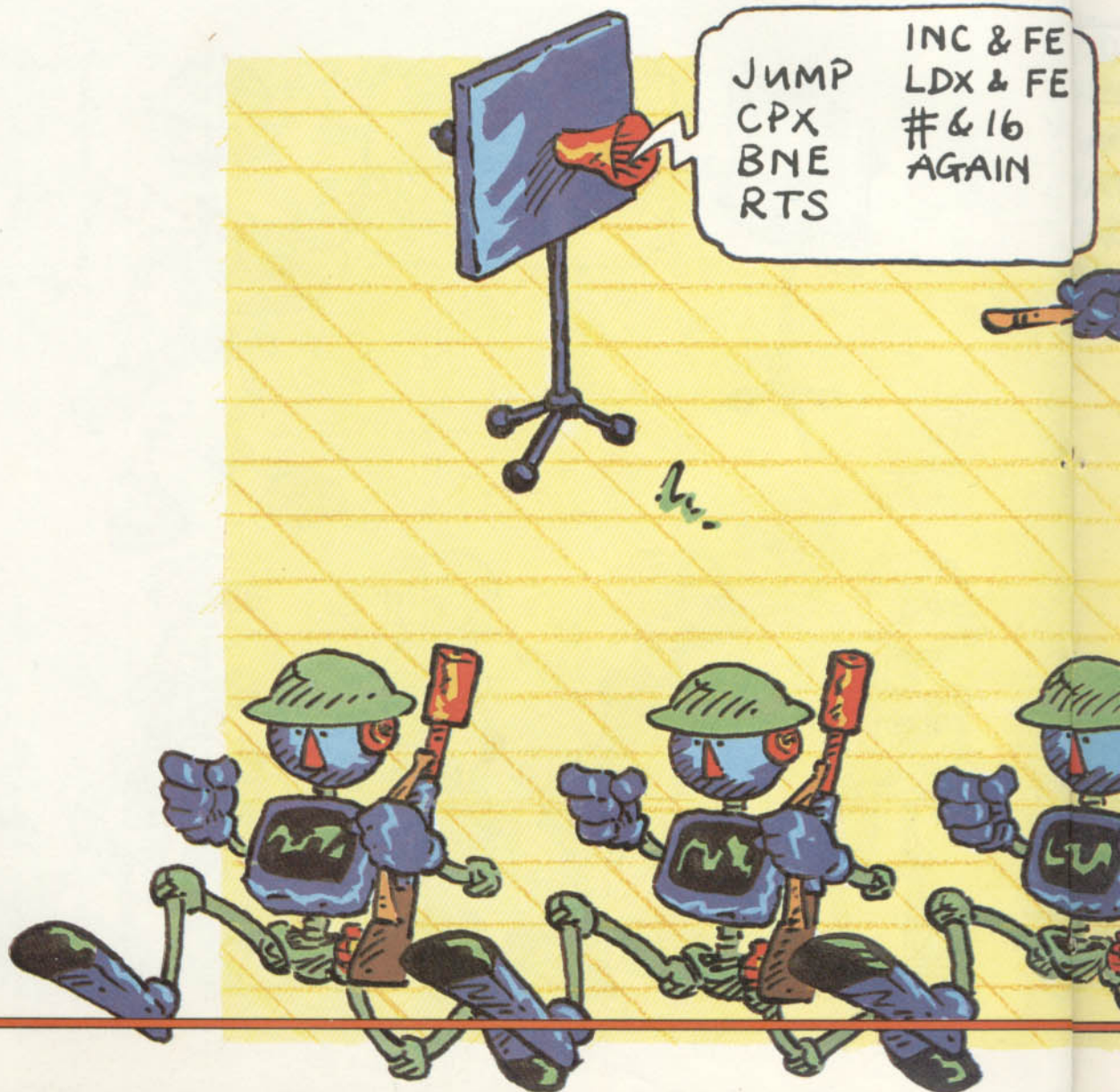
O comando **ld bc, 31** carrega os registros BC com o número 31. Cada linha tem 32 caracteres, mas a transferência do primeiro caractere para a última posição da linha é feita independentemente. Assim, basta contar até 31.

A instrução mais poderosa aqui utilizada é **ldir**, que significa "carregar, somar um e repetir" (=*load, increment and repeat*). Compreender o que ela faz é crucial para o entendimento do programa: o conteúdo da posição de memória cujo endereço está em HL — inicialmente, 16385 — é transferido para a posição cujo endereço está em DE — inicialmente, 16384. A seguir, deve-se somar 1 aos conteúdos de HL e DE, subtrair 1 do conteúdo de BC e verificar se o conteúdo de BC foi reduzido a 0. Enquanto isso não acontecer a instrução será repetida.

O que estava na segunda posição do vídeo passa então para a primeira; o que estava na terceira posição vai para a segunda, e assim até que termine a primeira linha do vídeo, o que acontece quando o registro de BC é reduzido a 0. A última transferência é a da 32ª posição, que passa para a 31ª. Nessa altura, o conteúdo de BC é subtraído de 1 e se torna 0, e o programa passa para a instrução seguinte.

O conteúdo do acumulador é transferido por **ld (de), a** para a posição cujo endereço está em DE. O acumulador contém o código do caractere que ocupava a primeira posição da memória de vídeo: 16384, endereço que estava em DE antes de **ldir** começar a incrementar esse registro.

Esse processo de somar 1 ao conteúdo de DE foi executado sucessivamente desde então, e agora DE contém o endereço da última posição da primeira linha. Este é justamente o local para onde desejamos transferir o caractere que estava no início da linha.

Os comandos **inc hl** e **inc de** somam

1 aos registros HL e DE, de forma que estes passam a conter os endereços das duas primeiras posições da linha seguinte. Como o conteúdo da primeira posição é transferido para o final da linha, independentemente de **ldir**, devemos incrementar o valor desses registros da mesma forma.

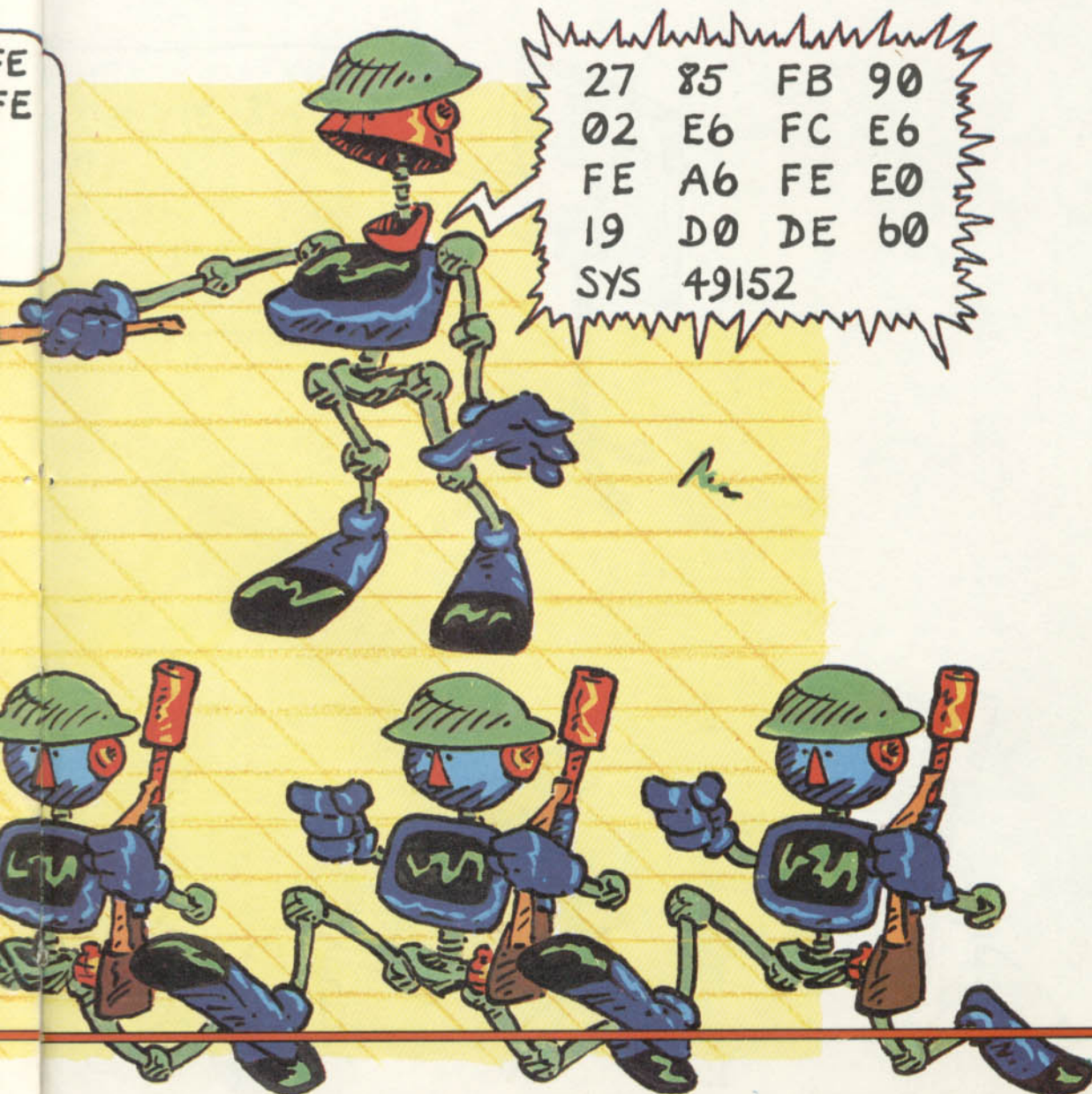
O antigo conteúdo de BC é recuperado da pilha pelo comando **pop bc**. O comando **djnz** subtrai 1 ao valor de B e "pula" sucessivamente para outros locais do programa, até que esse valor se iguale a 0 (= *decrement and jump if not zero*). No início do programa, carregamos o registro B com 192. Esse valor fica armazenado temporariamente na pilha para liberar o uso de BC. O comando **djnz** subtrai 1 a esse valor, que passa a ser 191 (valor diferente de 0). Assim, a instrução volta para a primeira ocorrência do rótulo **loop**.

Esse salto continua a ocorrer enquanto o conteúdo de B vai sendo reduzido de 192 até chegar a 0. O fato de esse valor ser transferido temporariamente para a pilha não altera o funcionamento das coisas. Desta forma, uma linha é deslocada de cada vez. Quando B finalmente contém o valor 0, após a última subtração feita por **djnz**, não ocorre salto, e o programa passa a instrução seguinte.

O comando **ret** significa retorne. Nenhuma rotina em código de máquina funcionará direito sem uma instrução desse tipo no final. O microprocessador tentará executar qualquer coisa que esteja após os códigos de seu programa. O conteúdo dessa parte da memória geralmente não faz sentido, e o mais comum é a perda de controle do computador e do conteúdo da memória.

Mesmo quando isso não ocorre, o microprocessador continua lendo e tentando executar tudo o que encontra até o final da memória. Chegando lá, ele retorna ao início da memória onde encontra as rotinas de inicialização do BASIC. Estas são executadas e a memória se apaga.

Tente traduzir para código de máquina e introduzir em seu micro a seguinte rotina Assembly.

```
     ld de,22527
     ld hl,22526
     ld b,192
loop push bc
     ld a,(de)
     ld bc,31
     lddr
     ld(de),a
     dec hl
     dec de
     pop bc
     djnz loop
     ret
```

Esse programa começa pelo final da memória de vídeo e vai transferindo os conteúdos das linhas "de trás para a frente". A diferença entre ele e o anterior é o uso da instrução **lddr**, que transfere o caractere da posição cujo endereço está em HL para a posição de endereço em DE, subtraindo 1 aos conteúdos de HL, DE e BC e repetindo o processo até que o valor em BC seja reduzido a 0. O comando **lddr** transfere blocos de memória em sentido contrário a **ldir**.

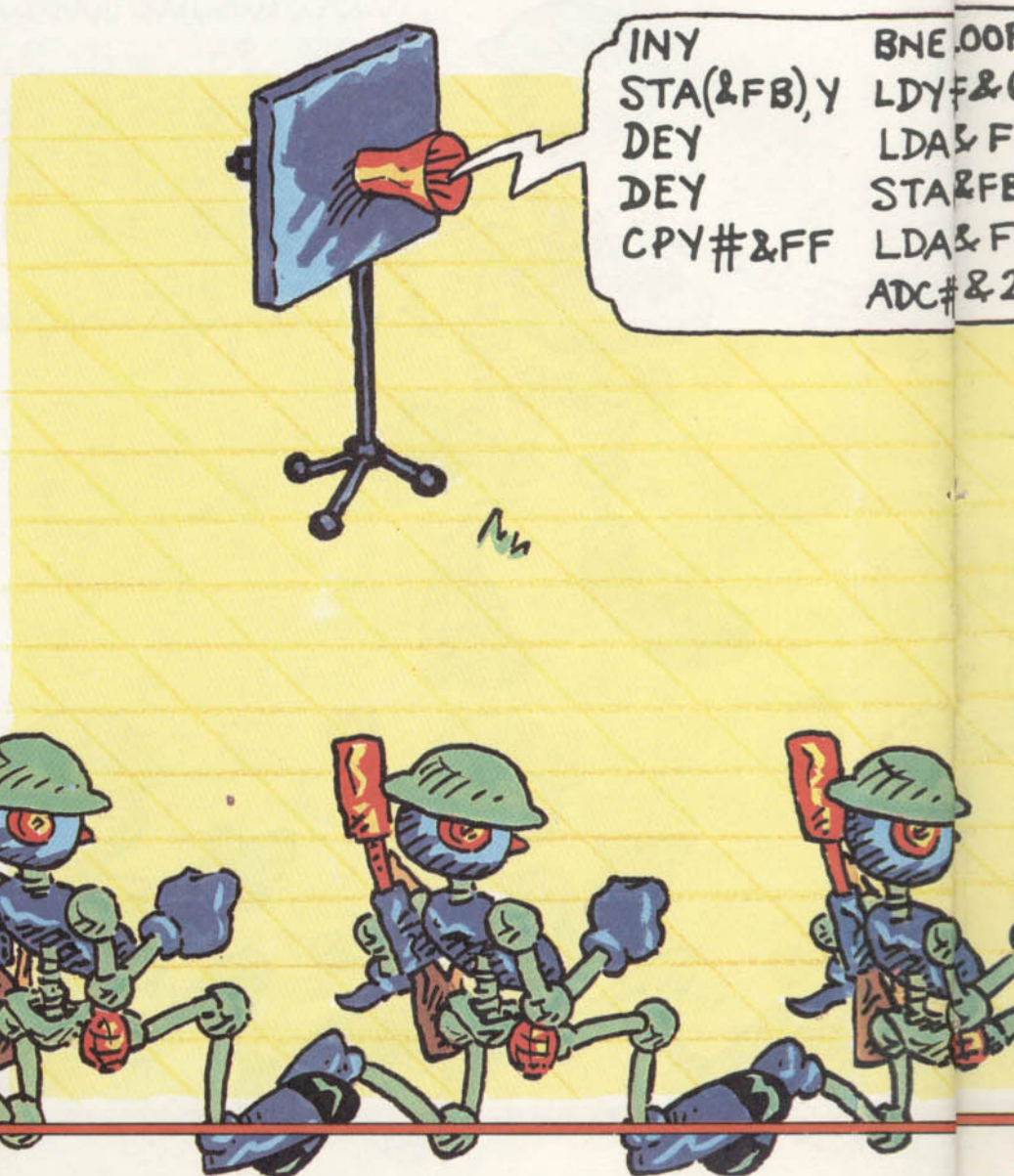
O salto é do mesmo tamanho que no programa anterior. Verifique se calculou seu valor corretamente, olhando a listagem (conte os bytes a partir do final da instrução de salto, ou seja, incluindo o byte cujo valor está sendo calculado).

O programa a seguir desloca a tela para a esquerda no ZX-81 com expansão de memória. Ele já está traduzido do Assembly para código de máquina.

```
     LD DE,(16396)  ED 5B 0C 40
     INC DE                  13
     LD H,D                  62
     LD L,E                  6B
     INC H,L                 23
     LD B,24              06 18
LOOP PUSH BC                 C5
     LD A,(DE)               1A
     LD BC,31          01 IF 00
     LDIR                 ED B0
     LD(DE),A                12
     INC HL                  23
     INC HL                  23
     INC DE                  13
     INC DE                  13
     POP BC                  C1
     DJNZ LOOP            10 F1
     RET                     C9
```

O PROGRAMA NO ZX-81

A memória de vídeo do ZX-81 não tem posição fixa; as posições 16396 e 16397 contêm o endereço da primeira posição do vídeo. A instrução **LDDE,(=16396)** carrega os registros D e E com os valores contidos em 16396 e 16397 — o conteúdo de 16396 vai para E, e o outro vai para D, seguindo a convenção "baixo-alto" do Z-80. Agora, o programa sabe onde está a memória de vídeo.

Esse micro conta com um caractere de retorno de carro (NEW LINE) no início de cada linha que não aparece na tela. Sua função é economizar o espaço ocupado pelo vídeo. Se a tela estiver vazia, a memória usada pelo vídeo corresponderá aos 24 caracteres NEW LINE, que marcam onde cada linha deve começar.

Quando é usada uma expansão, esses caracteres ficam no final de cada linha. Não devemos interferir nessa organização.

Tal organização é responsável pelas diferenças entre os programas do ZX-81 e os do Spectrum (leia com atenção a descrição do programa do Spectrum).

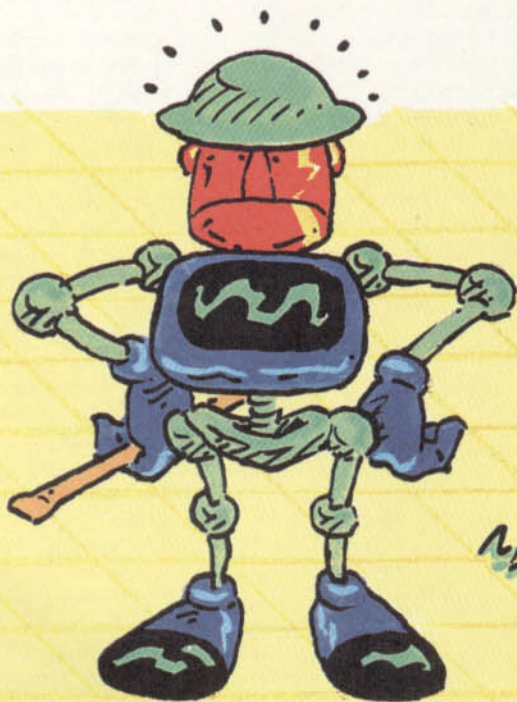
O primeiro caractere do vídeo é um NEW LINE. Como não devemos alterá-lo, a instrução **INC DE** soma 1 ao conteúdo de DE, que se torna então o endereço da segunda posição do vídeo.

As instruções **LD H,D** e **LD L,E** copiam o conteúdo do par DE no par de registros HL. **INC HL** soma 1 a HL, que passa a conter o endereço da terceira posição do vídeo. **LD B,24** carrega o registro B com o número de linhas da tela do ZX-81. O restante do programa funciona de maneira idêntica à do Spectrum. A única diferença é o número de comandos **INC HL** e **INC DE** que, no ZX-81, somam 1 a esses registros. Isso é feito uma vez para mover o programa para a linha seguinte e uma segunda vez para evitar interferências da linha.

Agora, tente obter os códigos hexadecimais correspondentes ao seguinte programa em Assembly, que desloca a tela para a direita.

```
     LD HL,(16396)
     LD DE,790
     ADD HL,DE
     LD D,H
     LD E,L
     INC DE
     LD B,24
LOOP PUSH BC
     LD A,(DE)
     LD BC,31
     LDDR
     LD(DE),A
     DEC HL
     DEC HL
     DEC DE
     DEC DE
     POP BC
     DJNZ LOOP
     RET
```

Esse exemplo é parecido com o segundo programa apresentado para o Spectrum. As diferenças entre eles são as mesmas que encontramos antes.

A organização da memória de vídeo do MSX é bem diferente da que existe em outras máquinas. Esse computador utiliza um microprocessador diferente da CPU para controlar a tela, dispondo de 16K que não estão na RAM para utilizar como memória de vídeo. Desta forma, essa porção da memória é tratada como dispositivo periférico, e o acesso a ela não é tão fácil quanto o acesso à RAM.

Isso significa que não podemos usar o comando **ld** para colocar ou ler um caractere na memória de vídeo. O comando Assembly utilizado para enviar dados a um dispositivo periférico é o **out**.

Quando a memória de vídeo de um computador está na RAM, podemos facilmente situar um caractere em qualquer posição vertical ou horizontal. Se a tela for tratada como um dispositivo periférico, contudo, a instrução **out** simplesmente colocará os caracteres na mesma ordem em que forem enviados, a partir da posição do cursor.

Temos, então, dois problemas: para controlar a posição do cursor, necessitamos de sub-rotinas da ROM; além disso, os 16K da memória de vídeo têm uma organização complicada, que varia conforme o tipo de tela selecionada, permitindo a exibição de caracteres, cores e sprites.

Enquanto não aprendemos como funcionam as sub-rotinas da ROM e a organização da memória de vídeo, vamos usar um artifício que dê ao MSX uma "memória de vídeo" na RAM, cujo acesso é bem mais fácil. Isso é feito pelas duas rotinas que se seguem.

A primeira copia os 960 caracteres da tela de textos de 40 colunas, no topo da memória, onde esta é manipulada de maneira semelhante à dos outros micros. A segunda copia os caracteres da "memória de vídeo" da RAM para a verdadeira memória de vídeo do MSX, que é um dispositivo periférico.

Embora todas as rotinas apresentadas funcionem independentemente de sua localização na memória, daremos exemplos utilizando endereços.

Proteja o topo da memória do MSX para colocar os programas e a "memória de vídeo" (= use **CLEAR 200, &HDFFF**). A seguir, carregue os programas usando o monitor. Utilize os endereços iniciais −8192 e −8155 para a primeira e a segunda rotinas, respectivamente.

```
      ld c,152              0E 98
      ld hl,-7936        21 00 E1
      ld a,4                3E 04
      ld b,240              06 F0
loop  inir                  ED B2
      dec a                    3D
      jrnz,loop             20 F9
      ret                      C9
```

```
      ld c,152              0E 98
      ld hl,-7935        21 01 E1
      ld a,4                3E 04
      ld b,240              06 F0
loop  outir                 ED B3
      dec a                    3D
      jrnz,loop             20 F9
      ret                      C9
```

Para fazê-las funcionar, digite ainda o seguinte programa em BASIC:

```
1000 SCREEN 0
1005 DEF USR1=-8192
1007 DEF USR3=-8155
1010 FOR I=0 TO 960
1020 VPOKE BASE(0)+I,RND(1)*256
1030 NEXT I
1040 VPOKE BASE(0),32:X=USR1(0)
1050 CLS:FOR I=1 TO 1000:NEXT I
1060 VPOKE BASE(0),32:X=USR3(0)
1080 GOTO 1080
```

Os possíveis erros podem ser corrigidos por meio do monitor. Para possibilitar isso, o programa em BASIC começa na linha 1000, acima do monitor. Se não apagarmos o programa monitor, poderemos rodar o novo programa com **RUN 1000**.

As duas rotinas fornecidas devem ser chamadas de um programa em BASIC. São especialmente importantes os comandos **VPOKE** das linhas 1020 e 1040, que posicionam o cursor no início da tela para que ela possa ser lida pela rotina **USR1** e escrita pela rotina **USR2**.

Agora, você pode montar a rotina de deslocamento horizontal propriamente dita. Ela é parecida com as rotinas dos outros micros que usam o Z-80. Não apague as duas rotinas anteriores. Utilize o endereço inicial −8112.

```
      ld de,-7935        11 01 E1
      ld hl,-7934        21 02 E1
      ld b,24               06 18
loop  push bc                  C5
      ld a,(de)                1A
      ld bc,39           01 27 00
      ldir                  ED B0
      ld (de),a                12
      inc de                   13
      inc hl                   23
      pop bc                   C1
      djnz loop             10 F3
      ret                      C9
```

Para vê-la funcionando, acrescente ainda as seguintes linhas ao programa em BASIC apresentado anteriormente:

```
1006 DEF USR2=-8112
1035 FOR I=1 TO 1000
1050 A=USR2(0)
1070 NEXT I
```

COMO FUNCIONAM OS PROGRAMAS

A primeira rotina transfere os 960 caracteres da tela de 40 colunas para o topo da memória. O comando **ld c,152** coloca o valor 152 no registro C. Este é o número da "porta" correspondente à memória de vídeo. Todo dispositivo periférico tem um número desse tipo.

Escolhemos os endereços de −7936 a −6976 (57600 a 58560; o computador não reconhece inteiros acima de 32768 em código de máquina) para colocarmos a nossa "memória de vídeo" na RAM. Assim, o primeiro endereço dessa memória é colocado no registro HL por **ldhl, −7936**.

O valor 4 é colocado no acumulador pelo comando **ld a,4**. O mesmo é feito com 240 no registro B, por **ld b,240**. A e B funcionam como contadores.

A instrução **inir** é a mais importante do programa. Ela coloca o valor lido na porta cujo número está em C na posição de memória cujo endereço está em HL. A seguir, a mesma instrução soma 1 ao conteúdo de HL e subtrai 1 a B. Se o conteúdo de B não for 0, todo o processo será repetido (*input, increment and repeat*). Quando o conteúdo de B se torna 0, o programa passa à instrução seguinte.

A instrução **dec a** subtrai 1 ao valor em A. Se, após essa operação, o conteúdo de A for diferente de 0, **jrnz** desviará o curso do programa para a primeira ocorrência do rótulo **loop** (*jump relative if not zero*). Isso fará com que o processo de ler 240 caracteres na porta 152 seja repetido. Caso o valor em A seja reduzido a zero, o salto não ocorrerá, e o programa passará então à instrução seguinte.

O comando **ret** faz com que a rotina retorne ao BASIC.

Em resumo, esse programa transfere os primeiros 960 caracteres da tela, a partir da posição do cursor.

Ele faz isto repetindo quatro vezes a leitura e transferência de 240 caracteres ($4 \times 240 = 960$).

A segunda rotina é essencialmente igual à primeira; apenas, trocamos o comando **inir** por **outir** (= *output, increment and repeat*) que, em vez de ler na porta 152, envia para esse dispositivo os 960 caracteres que estavam no topo da memória.

A terceira rotina é, com algumas modificações, a mesma utilizada para o Spectrum. Ela age sobre a "memória de vídeo" na RAM e, para funcionar, deve ser usada juntamente com as rotinas anteriores. Leia atentamente a explicação do programa do Spectrum.

As únicas diferenças são os endereços iniciais da memória de vídeo (−7935 e −7934), o número de linhas da tela (24) e o número de caracteres por linha, menos 1 (39). O restante é exatamente igual.

Tente então traduzir o programa que faz deslocamento horizontal da tela para a direita, do Assembly para código de máquina. Para rodá-lo, serão necessárias

as rotinas iniciais, bem como um programa em BASIC.

```
        ld de,-6976
        ld hl,-6977
        ld b,24
loop    push bc
        ld a,(de)
        ld bc,39
        lddr
        ld (de),a
        dec hl
        dec de
        pop bc
        djnz loop
        ret
```

O TRS-80 já vem com um monitor. Se seu sistema não tem unidade de disco, responda **N** à pergunta inicial do computador: BASIC (S/N)? Entramos, assim, no monitor. Para aprender os comandos específicos, consulte seu manual.

Caso seu sistema tenha unidade de disco, pressione simultaneamente **BREAK** e **RESET**, liberando a última tecla primeiro. Desse modo, o sistema se comportará como se não houvesse unidade de disco.

O programa montado com o monitor permanecerá na memória, não sendo apagado nem pelo DOS nem pela tecla **RESET**, nem pelo BASIC, se for dada uma resposta adequada à pergunta "Memória Usada?" (resposta adequada é um endereço inferior ao do programa montado).

A rotina que se segue desloca a tela um caractere para a esquerda.

```
        ld de,15360     11 00 3C
        ld hl,15361     21 01 3C
        ld b,16            06 10
loop    push bc               C5
        ld a,(de)             1A
        ld bc,63        01 3F 00
        ldir               ED B0
        ld (de),a             12
        inc de                13
        inc hl                23
        pop bc                C1
        djnz loop          10 F3
        ret                   C9
```

O PROGRAMA NO TRS-80

Este é essencialmente o mesmo programa fornecido para o Spectrum. Mas existem algumas diferenças.

São elas: o endereço inicial da memória de vídeo, que é 15360 no TRS; o número de linhas da tela (16) e o número de caracteres por linha (64). Os comandos são, portanto, idênticos.

A rotina a seguir desloca a tela para a direita.

```
        ld de,16383
        ld hl,16382
        ld b,16
loop    push bc
        ld a,(de)
        ld bc,63
        lddr
        ld (de),a
        dec hl
        dec de
        pop bc
        djnz loop
        ret
```

Dirija-se à seção do Spectrum para comentários.

Este programa desloca a tela para a esquerda. Já foi traduzido do Assembly para linguagem de máquina. Ele não funciona em sistemas com disquete.

```
        LDX#1024        8E 04 00
LOOP    LDB,X+             E6 80
        PSHS B             34 04
        LDB#31             C6 1F
JUMP    LDA,X+             A6 80
        STA-2,X            A7 1E
        DECB                  5A
        BNE JUMP           26 F9
        PULS B             35 04
        STB-1,X            E7 1F
        CMPX#1536       8C 06 00
        BLO LOOP           25 EA
        RTS                   39
```

COMO FUNCIONA O PROGRAMA

A memória de vídeo do computador começa no endereço 1024; assim, **LDX #1024** carrega este valor no registro X. **LDB,X +** carrega o acumulador B com o conteúdo da posição de memória cujo endereço está em X. A seguir, soma 1 ao conteúdo desse registro. **PSHB** armazena temporariamente na pilha o conteúdo do acumulador B — que é o código do primeiro caractere do vídeo.

O comando **LDB#31** carrega B com o número 31. Esse registro é usado como contador. Embora existam 32 caracteres por linha, a operação de deslocar cada caractere uma posição para a esquerda será feita 31 vezes. A transferência do primeiro caractere da linha para a última posição é feita separadamente, depois que o resto da linha é deslocado para a esquerda.

O comando **LDA,X +** carrega o acumulador A com o conteúdo do endereço que está em X; esse endereço está uma posição à frente do endereço carregado em B, pois foi somado 1 a X. Depois disso, a instrução soma 1 a X novamente. **STA – 2,X** armazena o conteúdo de A no endereço situado duas posições antes do conteúdo de X. Isso fica uma posição à esquerda de onde esse caractere foi retirado, já que X foi incrementado depois.

A instrução **DECB** subtrai 1 ao conteúdo de B, diminuindo esse valor de 31 até 0. Essa operação afeta o sinalizador 0 (ou *flag* zero), de forma que um resultado 0 em qualquer operação estabelece esse sinalizador. **BNE** verifica se o sinalizador foi estabelecido. Caso isto não tenha ocorrido, o curso do programa sofrerá um desvio. **JUMP** é o rótulo (*label*), de forma que o programa salta até a instrução **LDA,X +** e desloca para a esquerda o caractere seguinte, até chegar ao final da linha. Quando isso acontece, **DECB** reduz o conteúdo do registro B a 0 e o sinalizador é estabelecido. Agora, o salto não ocorre mais, e o programa passa à instrução seguinte a **BNE**.

O comando **PULS B** recupera o valor no topo da pilha, colocando-o em B. **STB – 1,X** faz com que o conteúdo de B seja armazenado uma posição atrás do endereço em X. Esta é a última posição da linha; o primeiro caractere é então recuperado da pilha. Assim, a linha é toda deslocada para a esquerda, e seu final passa a ser ocupado pelo primeiro caractere.

A instrução **CMPX#1536** compara o endereço em X com 1536, que é o último endereço da memória de vídeo. **BLO** desviará o programa se o conteúdo de X for menor que 1536 (*Branch if LOwer than*). Loop é o rótulo, de forma que, se o programa não deslocar toda a tela, chegando ao seu endereço final, o programa saltará para a instrução **LDB,X +** e começará a deslocar a linha seguinte.

Se o registro X contiver 1536, o programa terá deslocado toda a tela, e a instrução seguinte será executada.

O comando **RTS** retorna ao BASIC. Toda rotina em linguagem de máquina deve terminar com essa instrução. Se isso não ocorrer, o microprocessador tentará executar qualquer comando que esteja após o seu programa.

O programa seguinte desloca a tela para a direita. Traduza os mnemônicos Assembly para código de máquina (os endereços estão em decimal e devem ser convertidos para hexa).

```
        LDX#1536
LOOP    LDB,-X
        PSHS B
        LDB#31
JUMP    LDA,-X
        STA 1,X
        DECB
        BNE JUMP
        PULS B
        STB,X
```

```
        CMPX#1024
        BGT LOOP
        RTS
```

Este programa trabalha no sentido inverso ao anterior, começando pelo fim da memória de vídeo (1536). Ele desloca as linhas da tela para a direita, até chegar ao endereço 1024. Na realidade, 1536 é uma posição posterior ao final da tela, mas o pós-byte, – X, subtrai 1 ao conteúdo do registro X após a execução da instrução **LDX**. Assim, se **LDB, – X** carregar B com o conteúdo da última posição da tela, o programa deverá começar com um endereço que está uma posição na frente, dentro do registro X.

Com exceção desse detalhe, o programa funciona de maneira idêntica ao anterior (confira o tamanho dos saltos que calcular, consultando a listagem).

O Apple II já vem com um monitor. Para entrar, digite **CALL – 151**. Os comandos específicos estão no manual do computador. Para usar as rotinas em seus programas BASIC, basta acionar o comando **CALL N**, onde **N** é o endereço inicial da rotina em código. A organização da memória de vídeo do Apple II abriga duas páginas de vídeo com diferentes modos de utilização, além de uma página de texto. A maior complicação, porém, é a diferença na ordem em que as linhas de texto ou de pontos são representadas na tela e na memória. A segunda linha de texto na tela, por exemplo, não é o segundo conjunto de 40 bytes da memória de vídeo.

Assim, para não tornar muito complicado este primeiro programa em Assembly para o Apple II, fizemos com que ele deslocasse para a esquerda somente a primeira linha de vídeo.

Este programa não funcionará no TK-2000. Todavia, os usuários desse micro não precisam aprender a montar programas a mão, uma vez que dispõem do mini-Assembler. Use o endereço inicial hexa 320 (800 em decimal).

```
        LDA #$00            A9 00
        STA $FB             85 FB
        LDA #$04            A9 04
        STA $FC             85 FC
        LDA #$00            A9 00
        STA $FE             85 FE
        LDY #$00            A0 00
        LDA ($FB),Y         B1 FB
        STA $FD             85 FD
        LDY #$01            A0 01
LOOP    LDA ($FB),Y         B1 FB
        DEY                    88
        STA ($FB),Y         91 FB
        INY                    C8
        INY                    C8
        CPY #$28            C0 28
        BNE LOOP            D0 F5
        LDY #$27            A0 27
        LDA $FD             A5 FD
        STA ($FB),Y         91 FB
        RET                    60
```

Use ainda este programa em BASIC:

```
5   HOME
10   FOR I = 1 TO 40
20   VTAB 1: PRINT  CHR$ (64 + I
);
30   NEXT
40   FOR I = 1 TO 1000
50   CALL 800
60   NEXT
70   GOTO 70
```

O PROGRAMA NO APPLE II

O comando **LDA #$00** carrega 0 no acumulador, e **STA $FB** coloca esse valor no endereço 0020. Analogamente, **LDA #$04** e **STA $FC** colocam 04 em 00FC. Não há um comando único que armazene valores diretamente na memória.

O endereço da primeira posição da memória de vídeo do Apple é 0400 (em hexa). 00FB e 00FC são posições da página 0 da memória. As posições dessa página requerem somente um byte no seu endereçamento. **LDA #$00** e **STA $FE** colocam 0 em 00FE, posição que será usada como contador.

A instrução **LDY #$00** carrega o registro-índice Y com o deslocamento (ou seja, 0, pois o programa tem que começar no início da tela), **LDA (FB), Y** carrega o acumulador com o conteúdo da posição de memória cujo endereço está em 00FB e 00FC, mais um deslocamento provocado pelo número que está em Y. 00FB e 00FC apontam para a posição 0400, início da tela de textos. Assim, essa instrução posiciona o primeiro caractere no acumulador. **STA $FD** o coloca na posição 00FD.

A seguir, **LDY #$01** carrega o registro Y com 1. **LDA ($FB),Y** carrega o acumulador com o conteúdo da posição de endereço dado por 00FB e 00FC, mais um deslocamento provocado por Y. Desta vez, contudo, Y contém 1 em vez de 0. Assim, essa instrução põe no acumulador o caractere da posição 0401.

A instrução **DEY** subtrai 1 ao registro Y, e **STA ($FB),Y** coloca o caractere cujo código está em A na posição dada por 00FB e 00FC, mais o deslocamento em Y. Como Y foi diminuído de 1 nesse processo, ele tem como resultado a transferência de cada caractere para uma posição à esquerda.

O registro Y tem 1 duas vezes somado ao seu conteúdo por intermédio da repetição do comando **INY**, apontando assim para a próxima posição da tela **CPY #$28** compara o conteúdo de Y com 28 em hexa, ou 40 em decimal, que é o total de caracteres por linha. **BNE LOOP** verifica o sinalizador 0. Se ele não for estabelecido, o microprocessador voltará para o endereço rotulado por **LOOP** e continuará de lá.

O comando **BNE** envia o programa de volta a **LOOP** até que Y acabe de contar de 0 a 40, movendo os caracteres da linha para a esquerda. Quando o conteúdo de Y chega a 40, a condição de **BNE** não é satisfeita e o microprocessador continua na instrução seguinte.

A instrução **LDY #$27** carrega Y com 27 hexa, ou 39 decimal. **LDA $FD** carrega o acumulador com o conteúdo da posição de memória 00FD.

Já o comando **STA ($FB),Y** coloca o conteúdo do acumulador no endereço dado por 00FB e 00FC, mais o deslocamento em Y. Ele posiciona o primeiro caractere da linha na posição 0427, que é a última da primeira linha de texto.

A instrução **RTS** diz ao microprocessador para voltar ao BASIC. Qualquer rotina em linguagem de máquina tem que terminar com **RTS**, caso contrário o microprocessador seguirá memória afora, tentando executar qualquer coisa que houver no caminho. O programa seguinte desloca a primeira linha de vídeo para a direita. Tente calcular os códigos correspondentes.

```
        LDA #$00
        STA $FB
        LDA #$04
        STA $FC
        LDA #$00
        STA $FE
        LDY #$27
        LDA ($FB),Y
        STA $FD
        LDY #$26
LOOP    LDA ($FB),Y
        INY
        STA ($FB),Y
        DEY
        DEY
        CPY #$FF
        BNE LOOP
        LDY #$00
        LDA $FD
        STA ($FB),Y
        RET
```

Este programa opera da mesma forma que o anterior, exceto que somamos onde subtraímos e vice-versa, para que o deslocamento seja feito no sentido inverso.

Os saltos são do mesmo tamanho, de forma que os valores calculados podem ser verificados na outra listagem (o salto inclui o final da instrução **BNE**, ou seja, o byte que contém seu tamanho).

COMO ESTRUTURAR SEUS PROGRAMAS

- O QUE É PROGRAMAÇÃO ESTRUTURADA?
- OS FLUXOGRAMAS
- COMO ESTRUTURAR PROGRAMAS EM BASIC

A estruturação de um programa é uma técnica que facilita a compreensão de seu funcionamento e permite tornar mais eficiente e simples sua operação. Aprenda a utilizá-la em BASIC.

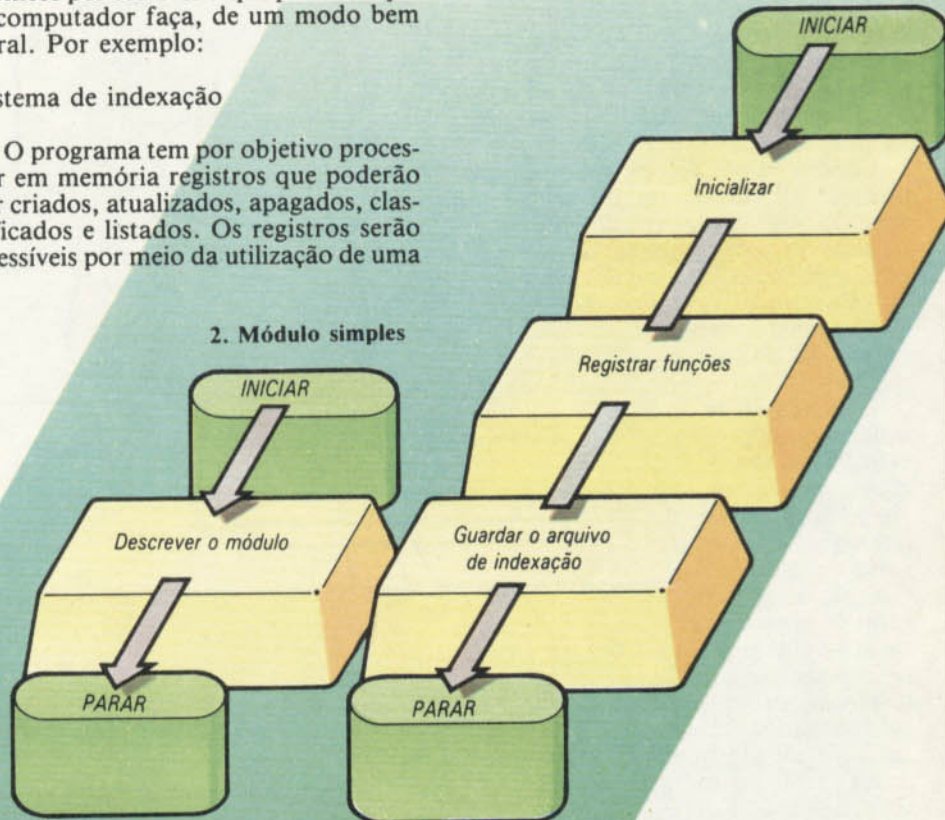
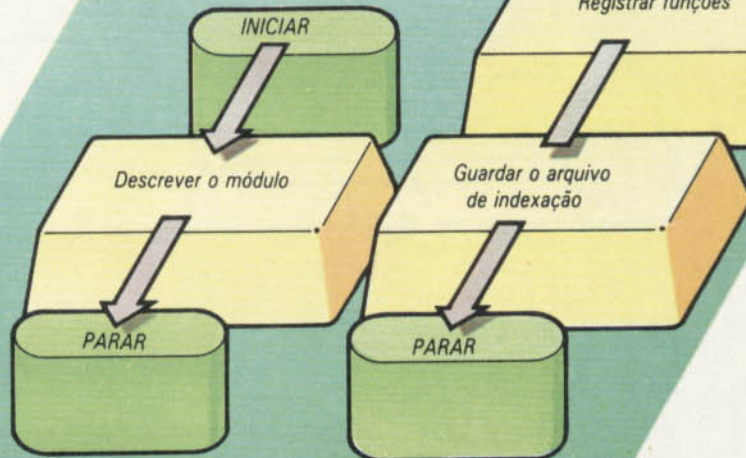
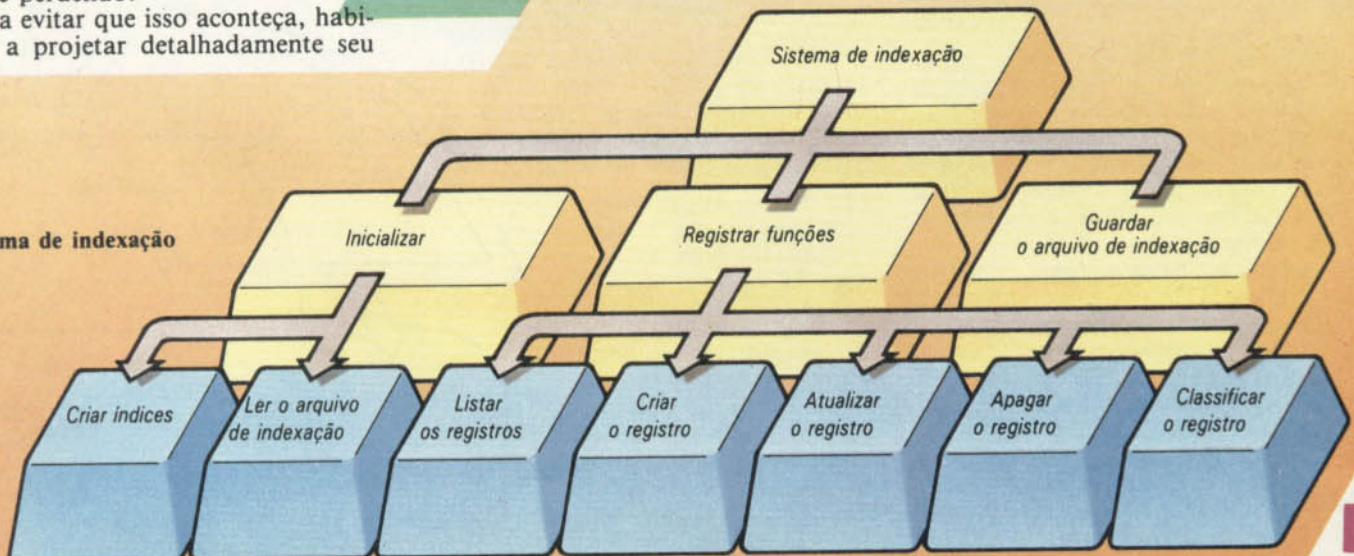
O interpretador BASIC padrão, disponível na maioria dos micros, é simples de usar, mas possui poucas *estruturas de programação*, o que dificulta a elaboração de programas estruturados.

As estruturas são "blocos de construção" da lógica de um programa, e são utilizadas para proporcionar-lhe uma forma coerente e de fácil compreensão. No BASIC padrão, as principais estruturas de programação lógica são representadas pelas declarações **IF...THEN**, **FOR...NEXT**, **GOTO** e **GOSUB**. Já vimos como empregar individualmente a maioria delas; examinaremos agora como colocá-las juntas de uma maneira legível e ordenada.

Em programas curtos, a estruturação não é uma questão muito importante. O problema aparece quando se pretende elaborar um programa longo e complexo, do ponto de vista lógico (desvios). Nesse caso, se você escrever uma parte do programa e depois acrescentar mais e mais coisas a ele, provavelmente acabará se perdendo.

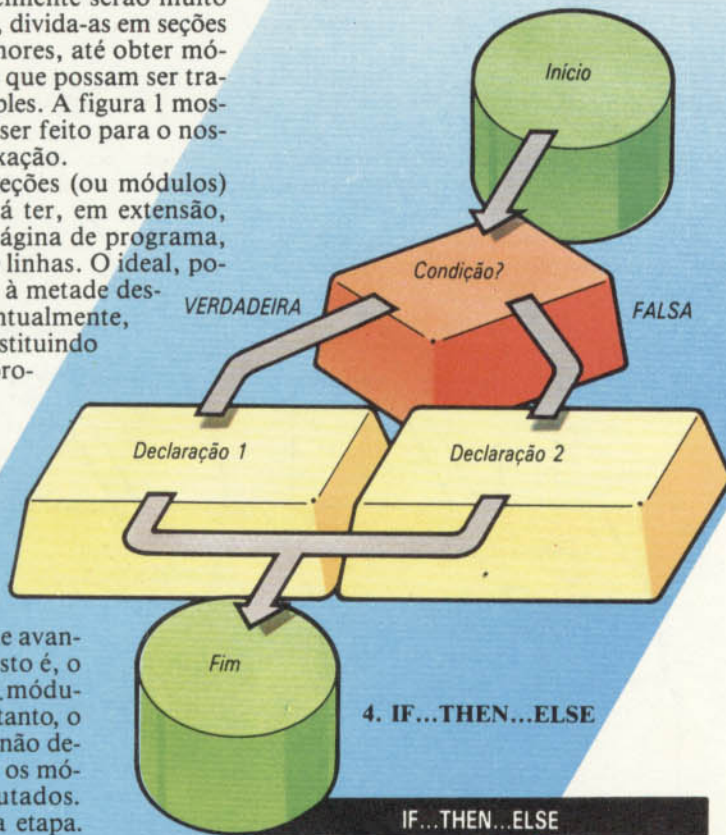
Para evitar que isso aconteça, habitue-se a projetar detalhadamente seu programa, antes de começar a digitá-lo. Comece por escrever o que pretende que o computador faça, de um modo bem geral. Por exemplo:

Sistema de indexação

O programa tem por objetivo processar em memória registros que poderão ser criados, atualizados, apagados, classificados e listados. Os registros serão acessíveis por meio da utilização de uma

palavra-chave. Todo o conjunto de registros poderá ser armazenado e carregado em um arquivo.

Este primeiro passo é conhecido como especificação do sistema. O passo seguinte consistirá em dividir as especificações gerais, acima, em etapas lógicas ou módulos. Estes, por sua vez, também terão suas funções especificadas de maneira geral. As operações envolvidas em cada módulo provavelmente serão muito complicadas; assim, divida-as em seções sucessivamente menores, até obter módulos tão pequenos que possam ser tratados de forma simples. A figura 1 mostra como isso pode ser feito para o nosso sistema de indexação.

Cada uma das seções (ou módulos) menores não deverá ter, em extensão, mais do que uma página de programa, ou seja, cerca de 60 linhas. O ideal, porém, será limitá-las à metade deste tamanho. Eventualmente, elas terminarão constituindo sub-rotinas do seu programa.

Esse método de subdividir um problema é conhecido como *modelo de cima para baixo (top-down)*. Começa-se do alto (ou seja, da descrição geral do programa) e avança-se até embaixo (isto é, o nível mais baixo de módulos). Até aqui, entretanto, o programador ainda não decidiu em que ordem os módulos serão executados. Esta será a próxima etapa.

DIAGRAMAS DE BLOCOS

A ordem (ou lógica) de execução dos módulos de um programa pode ser especificada por meio dos *diagramas de blocos* ou *fluxogramas*. O uso destes permite que se ordene o programa de uma maneira bem estruturada e clara. Para isto, basta seguir um conjunto simples de regras.

A figura 2 mostra como especificar um módulo. O programa acompanha as linhas na direção das flechas e os retângulos descrevem o que acontece em cada estágio. Se quiser executar uma série de módulos em seqüência, acrescente mais retângulos na ordem correta, entre o início e o fim (veja a figura 3).

Um fluxograma como este é ideal para um programa no qual nenhuma decisão é tomada. No entanto, o poder do computador reside exatamente em sua habilidade de fazer escolhas. A familiar declaração **IF...THEN** é a estrutura mais freqüentemente utilizada para isso, na linguagem BASIC. Vamos agora examinar sua representação e a de outros tipos de estrutura em um fluxograma.

4. IF...THEN...ELSE

IF...THEN...ELSE

Embora os diversos computadores utilizem de maneira diferente a declaração **IF...THEN...ELSE**, ela é sempre a base da tomada de decisões dentro de um programa. Representada no fluxograma por um losango (veja figura 4), esta declaração é escrita em BASIC da seguinte maneira:

```
100 IF condição THEN
    declaração 1 ELSE
    declaração 2.
```

Ou seja: se a condição for verdadeira, será executada a declaração 1; caso contrário, a declaração 2.

Note que, no fluxograma, só há um ponto de saída para a estrutura **IF...THEN...ELSE**. De fato, cada seção de código deve ter apenas uma entrada e uma saída. Esta regra, muito importante para a programação estruturada, auxilia bastante a testar e a eliminar as falhas.

Nem todas as versões em BASIC, porém, possuem a partícula **ELSE** — ela não está disponível, por exemplo, nos computadores Spectrum, ZX-81, Apple ou TK-2000. Nesses casos, para evitar que a estrutura passe a ter mais de uma saída, o **ELSE** pode ser simulado utilizando-se o **GOTO**:

```
100 ...
110 IF condição THEN GOTO 140
120 declaração 2
130 GOTO 150
140 declaração 1
150 ...
```

É claro que os números das linhas não precisam ser iguais aos do exemplo. Além disso, pode-se incluir mais de uma declaração nas partículas **THEN** e **ELSE**. Veja, a seguir, uma seção de programa para classificar dois números dentro de determinada ordem. Ela constitui a base para uma rotina de classificação alfabética que será dada adiante. No caso específico, a partícula **ELSE** possui quatro declarações:

```
100 IF  primeiro<=segundo  THEN
    GOTO 160
110 LET temporario=primeiro
120 LET primeiro=segundo
130 LET segundo=temporario
140 LET ordem$="errado"
```


5. IF...THEN

```
150 GOTO 170
160 LET ordem$="certo"
170 ...
```

Esta seção também pode ser escrita utilizando-se o **ELSE**, mas ele deverá estar totalmente contido em apenas uma linha. As declarações múltiplas serão permitidas desde que estejam separadas por dois pontos:

```
100 IF  primeiro>segundo   THEN
    temporario=primeiro        :
    primeiro=segundo:  segundo=
    temporario:ordem$="errado"
    ELSE ordem$="certo"
```

Não é fácil ler ou entender programas que utilizam declarações longas como esta — assim, sempre que possível, deve-se evitá-las.

Finalmente, em alguns casos, não será necessário usar partícula **ELSE**. O fluxograma se parecerá com o da figura 5 e será redigido desta maneira:

```
100 IF  condição THEN  declara-
    ção.
```

que é, simplesmente, a própria declaração **IF...THEN**.

ESTRUTURAS EMBUTIDAS

É possível embutir uma linha **IF... THEN...ELSE** dentro de outra declaração **IF...THEN...ELSE**. Ou seja, a conseqüência de um **THEN** ou de um **ELSE** também poderá ser um outro **IF**, e assim por diante. É o caso da seção de programa abaixo, que executa a contagem de quantas partidas dois jogadores ganharam e imprime os resultados após cada jogo:

```
100 IF T1<>T2 THEN GOTO 130
110 PRINT"EMPATOU!"
120 GOTO 190
130 IF T1<T2 THEN GOTO 170
140 PRINT"O JOGADOR 1 GANHA"
150 LET P1=P1+1
160 GOTO 190
170 PRINT"O JOGADOR 2 GANHA"
180 LET P2=P2+1
190 ...
```

Todas as estruturas podem ser embutidas em qualquer combinação e, na teoria, em qualquer profundidade. No entanto, quanto mais você as embute, menos legível se torna o programa; assim, é razoável estabelecer um limite de profundidade de três ou quatro estruturas. Se precisar embutir mais estruturas, subdivida o programa em módulos ou subrotinas menores.

Examinando o último programa, você observará que é difícil acompanhá-lo, ainda que esteja perfeitamente estruturado. Uma maneira prática de tornar mais legíveis as declarações embutidas é recuar (indentar) as linhas do programa. Isso só é viável nos computadores das linhas TRS-80, TRS-Color, Spectrum e MSX; neles, você poderá reescrever o último programa da forma que se segue:

```
100 IF T1<>T2 THEN GOTO 130
110     PRINT"EMPATOU!"
120 GOTO 190
130 IF T1<T2 THEN GOTO 170
140     PRINT"O JOGADOR 1 GANHA"
150     LET P1=P1+1
160 GOTO 190
170     PRINT"O JOGADOR 2 GANHA"
180     LET P2=P2+1
190 ...
```

A introdução de linhas em branco em diferentes seções e a utilização de declarações **REM** constituem outras maneiras de tornar mais clara a estrutura de um programa. Para entrar as linhas em branco nos computadores Spectrum, basta digitar o número da linha seguida de um espaço e, em seguida, pressionar **<ENTER>** ou **<RETURN>**. Ao escrever programas no TRS-80, TRS-Color ou MSX, você poderá obter um efeito semelhante digitando um apóstrofo ('), ao invés de dar o espaço.

WHILE...DO

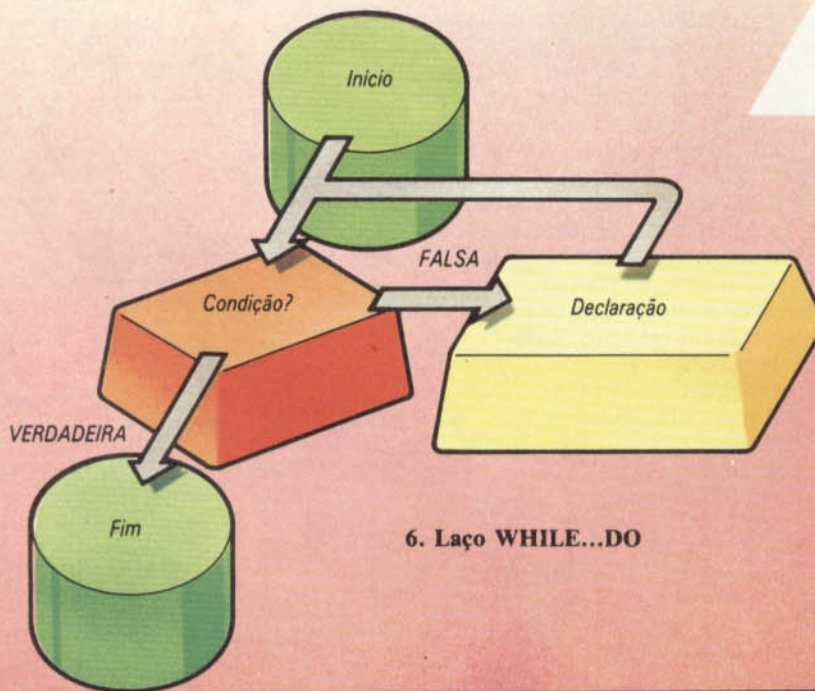
Outra declaração importante em programação estruturada é a **WHILE...DO**, que permite a repetição em um programa e é um dos recursos mais úteis para se criar um laço. Esta declaração diz ao computador que faça (**DO**) repetidamente uma coisa enquanto (**WHILE**) uma certa condição for verdadeira. Se seu computador não possui as palavras **WHILE** e **DO**, você pode inventar uma estrutura que faça a mesma coisa, utilizando **IF...THEN** e **GOTO**. A redação, em BASIC, é a seguinte (veja o fluxograma correspondente na figura 6):

```
100 ...
110 IF  NOT condição THEN  GOTO
    140
120 declaração
130 GOTO 110
140 ...
```

Note que a linha 100 lê a condição **IF NOT...**, ou seja, verifica se a condição não é verdadeira. O procedimento é oposto do normal, ao qual você está acostumado, mas não tem problema. Se a condição for **A=B** então **NOT (A=B)**, será equivalente a **A< >B**. Do mesmo modo, **NOT (A<B)** será **A>=B** e assim por diante. Na verdade, você poderá escrever **NOT (A=B)** se quiser, e o computador entenderá o que você quer dizer.

Eis aqui o exemplo de um programa curto (para cronometrar o cozimento de um ovo) que utiliza um laço **WHILE**:

6. Laço WHILE...DO

```
 5 CLS
10 PRINT AT 3,11;"CONTADOR"
20 INPUT "Quantos minutos voc
e quer?",t
30 PRINT AT 7,5;"Pressione qu
alquer tecla        para come
car"
40 PAUSE 0
50 CLS
60 PRINT  FLASH 1;AT 10,10;"
T E M P O "
70 POKE 23672,0: POKE 23673,0
```

```
80 LET time=PEEK 23672+256*
PEEK 23673: IF time>t*50*60
THEN  GOTO 110
90 PRINT AT 14,10;INT (time/
50);" segundos"
100 GOTO 80
105 REM fim do loop WHILE
110 PRINT  FLASH 1;AT 14,10;"
TERMINADO! "
120 SOUND .5,20
```

```
5 CLS
10 LOCATE 10,3:PRINT"CONTADOR D
E TEMPO"
20 LOCATE 0,6:INPUT"QUANTOS MIN
UTOS VOCÊ QUER";T
```

7. Escolhas múltiplas

```
30 PRINT:PRINT"
PRESSIONE QU
ALQUER TECLA
PARA COMEÇAR";
40 IF INKEY$=
""THEN 40
50 CLS
60 LOCATE 10,15:PRINT"CONTANDO.
.."
70 TIME=0
75 REM COMEÇA O LOOP WHILE
80 IF TIME>T*3600 THEN GOTO 110
90 LOCATE 12,17:PRINTINT(TIME/6
0);"seg."
100 GOTO 80
105 REM FIM DO LOOP WHILE
110 LOCATE 10,19:PRINT"ESTÁ PRO
NTO!"
120 BEEP
```

```
5 CLS
10 PRINT @43,"CONTADOR"
15 LET E$=CHR$(27)
20 PRINT @129,;:INPUT"QUANTOS M
INUTOS VOCE QUER ?";T
30 PRINT @196,"PRESSIONE QUALQU
ER TECLA        PARA COMECAR"
40 A$=INKEY$:IF A$="" THEN GOTO
 40
50 CLS
60 PRINT @238,"TEMPO"
70 TIMER = 0
75 REM COMECO DO LACO "WHILE"
80 IF TIMER>T*3000 THEN GOTO 11
0
90 PRINT @298,INT(TIMER/50);"SE
GUNDOS"
100 GOTO 80
105 REM FIM DO LACO "WHILE"
110 PRINT @364,"TERMINADO!"
120 SOUND 180,3
```

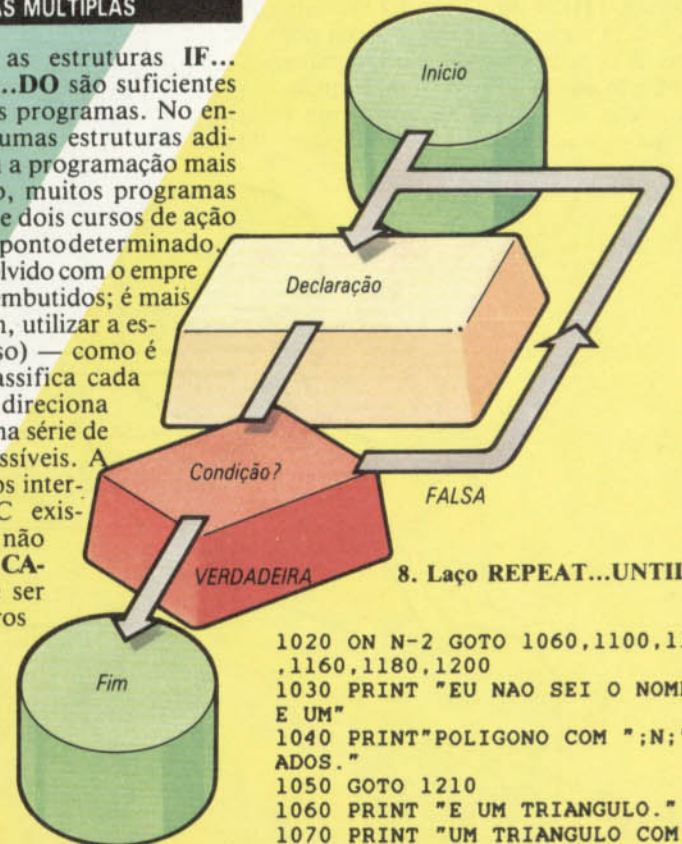
ESCOLHAS MÚLTIPLAS

Normalmente, as estruturas **IF... THEN** e **WHILE...DO** são suficientes para a maioria dos programas. No entanto, existem algumas estruturas adicionais que tornam a programação mais fácil. Por exemplo, muitos programas apresentam mais de dois cursos de ação possíveis para um ponto determinado. Isso poderia ser resolvido com o emprego de **IF...THEN** embutidos; é mais conveniente, porém, utilizar a estrutura **CASE** (caso) — como é conhecida. Ela classifica cada opção em ordem e direciona o computador a uma série de cursos de ação possíveis. A grande maioria dos interpretadores BASIC existentes para micros não inclui a declaração **CASE**, mas ela pode ser simulada por outros comandos padrão.

O fluxograma para uma estrutura **CASE** é mostrado na figura 7 e, em BASIC, um programa típico seria:

```
100 REM COMANDO CASE
110 IF C$="C" THEN GOTO 170
120 IF C$="M" THEN GOTO 190
130 IF C$="A" THEN GOTO 210
140 IF C$="L" THEN GOTO 230
150 PRINT "comando nao reconhec
ido"
160 GOTO 240
170 PRINT "CRIAR REGISTRO":GOSU
B 1000
180 GOTO 240
190 PRINT "MODIFICAR REGISTRO":
GOSUB 2000
200 GOTO 240
210 PRINT "APAGAR REGISTRO":GOS
UB 3000
220 GOTO 240
230 PRINT "LISTAR REGISTROS":GO
SUB 4000
240 REM FIM DO CASE
```

Pode-se também fazer múltiplas escolhas por meio das declarações **ON...GOTO** e **ON...GOSUB**. Entretanto, quando utilizar **ON...GOTO**, verifique se cada uma das opções direciona você ao final da seção. Na sub-rotina abaixo, por exemplo, foi necessário usar um **GOTO 1210** após cada opção, para direcionar o programa para o fim da rotina. Com uma declaração **CASE** tal procedimento seria desnecessário.

```
1000 REM SUBROTINA PARA POLIGON
OS
1010 INPUT"QUANTOS LADOS VOCE Q
UER?";N
```

8. Laço REPEAT...UNTIL

```
1020 ON N-2 GOTO 1060,1100,1140
,1160,1180,1200
1030 PRINT "EU NAO SEI O NOME D
E UM"
1040 PRINT"POLIGONO COM ";N;" L
ADOS."
1050 GOTO 1210
1060 PRINT "E UM TRIANGULO."
1070 PRINT "UM TRIANGULO COM OS
```

```
LADOS IGUAIS CHAMA-SE"
1080 PRINT "TRIANGULO EQUILATER
O."
1090 GOTO 1210
1100 PRINT "E UM QUADRILATERO."
1110 PRINT "UM QUADRILATERO COM
 OS LADOS"
1120 PRINT "E ANGULOS IGUAIS CH
AMA-SE QUADRADO."
1130 GOTO 1210
1140 PRINT "E UM PENTAGONO."
1150 GOTO 1210
1160 PRINT "E UM HEXAGONO."
1170 GOTO 1210
1180 PRINT "E UM HEPTAGONO."
1190 GOTO 1210
1200 PRINT "E UM OCTOGONO."
1210 PRINT
1220 RETURN
```

Como se trata de uma sub-rotina, você ainda não poderá rodá-la. O programa para chamar a rotina será dado na próxima seção.

REPEAT...UNTIL

A declaração **REPEAT... UNTIL** também é uma estrutura muito útil para fazer o programa repetir um grupo de comandos (se seu computador não possui esses comandos, poderá simulá-los por meio de outros). Nesse caso, ao contrário do **WHILE...DO**, o laço sempre é executado pelo menos uma vez. Veja o símbolo correspondente no fluxograma da figura 8 e compare-o à figura 6. Em BASIC, será escrito assim:

```
110 declaração
120 IF  NOT condição THEN  GOTO
    110
130 ...
```

Utilizando a sub-rotina do último exemplo, você poderá escrever um programa utilizando um laço do tipo **REPEAT**, como o seguinte:

```
10 PRINT "VOU DIZER OS NOMES"
20 PRINT "DE ALGUNS POLIGONOS."
30 REM COMECO DO LACO
40 GOSUB 1000
50 INPUT "VOCE QUER OUTRO NOME
?";A$
60 IF LEFT$(A$,1)="S" THEN GOTO
 30
70 PRINT "TCHAU !":END
```

A linha 1000 é a sub-rotina de polígono dada na seção anterior.

LAÇOS FOR...NEXT

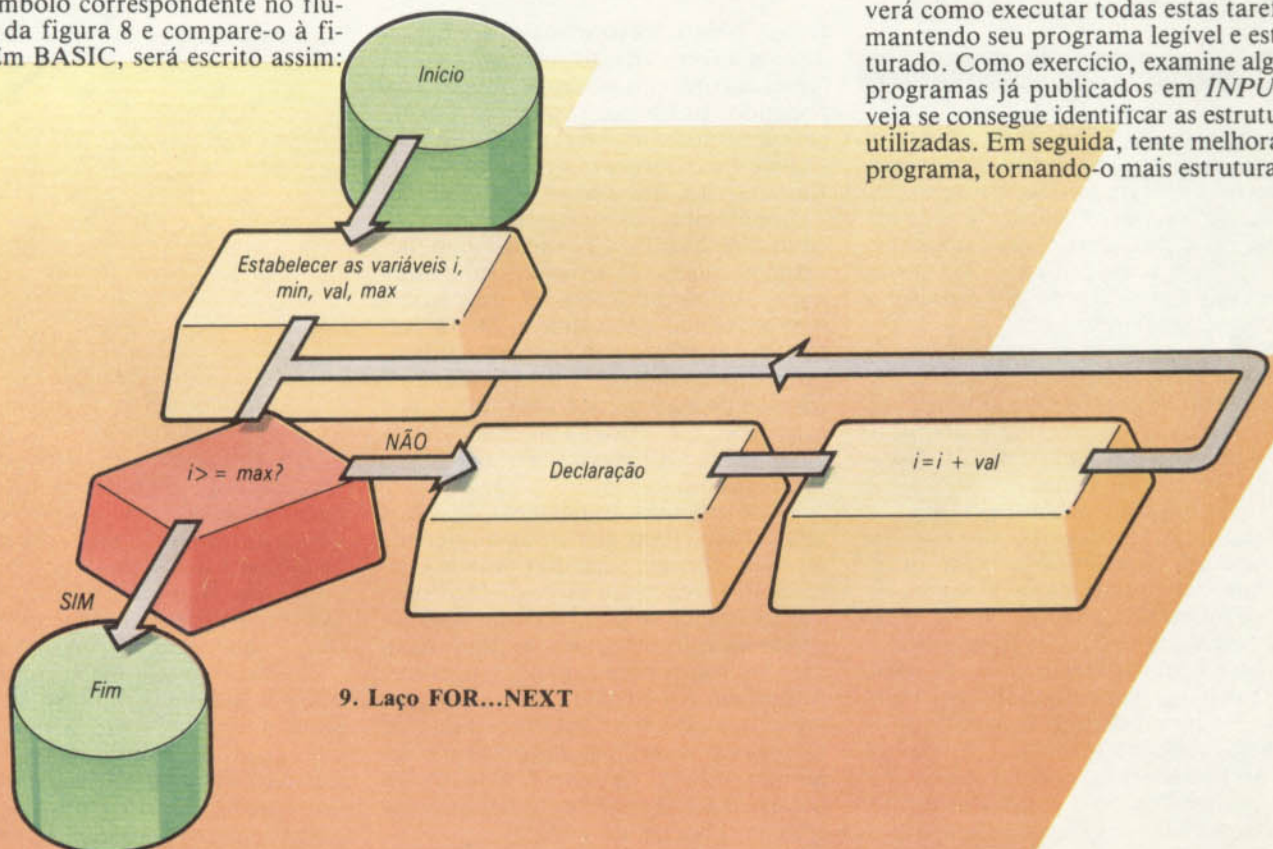
Talvez você não tenha percebido, mas o familiar laço **FOR...NEXT** é apenas um caso especial do laço **WHILE...DO**. Ele pode ser utilizado quando se conhece antecipadamente o número de vezes que o laço deverá ser repetido, pois esta informação deve ser especificada logo no início. A variável que mantém a contagem do número de vezes ao redor do laço é conhecida como *variável de controle*.

Um fluxograma para o laço **FOR...NEXT** é semelhante ao que se vê na figura 9. Comparando-o com o laço **WHILE** da figura 6, observa-se que tem a mesma estrutura geral. Em BASIC é descrito assim:

```
100 FOR i=min TO max STEP val
110 declaração
120 NEXT i
```

Não se deve saltar para fora de um laço **FOR**, utilizando uma declaração **GOTO**, já que o interpretador BASIC não tem como identificar o que foi feito. Será fácil saltar de volta dentro de um laço, no programa, mas a compreensão de tal estrutura é muito difícil. Assim, evite este procedimento.

COMO COLOCAR TUDO JUNTO

As estruturas vistas neste artigo são as mais conhecidas e importantes, e podem ser utilizadas em qualquer tipo de programa. Mas, para completár o programa, você deverá ainda especificar as variáveis que irá utilizar e verificar se as variáveis dos diferentes módulos não entrarão em choque umas com as outras. Além disso, será preciso especificar todas as entradas e saídas necessárias, testar cada um dos módulos e encadeá-los todos juntos. No próximo artigo você verá como executar todas estas tarefas, mantendo seu programa legível e estruturado. Como exercício, examine alguns programas já publicados em *INPUT* e veja se consegue identificar as estruturas utilizadas. Em seguida, tente melhorar o programa, tornando-o mais estruturado.

9. Laço FOR...NEXT

O MAPA DA AVENTURA

Acompanhe a criação de uma aventura e, depois, crie seu próprio jogo. Aqui, os primeiros passos: o desenvolvimento do roteiro e a confecção do mapa dos locais.

Antes de programar um jogo, é importante elaborar minuciosamente o roteiro, para evitar dificuldades posteriores, como erros e pontos obscuros a serem esclarecidos.

Para que você se familiarize com esta etapa do trabalho, acompanhe conosco o desenvolvimento de um programa de aventuras típico. A história que tomamos como exemplo se passa em um país distante, onde o jogador deve encontrar o olho perdido de um legendário totem inca. Se você seguir passo a passo as etapas percorridas para a montagem desta aventura, sentirá maior facilidade ao projetar seu próprio jogo.

O ROTEIRO

Em primeiro lugar o programador precisa criar um "mundo" que combine com a estrutura básica do roteiro. Este mundo compõe-se de uma série de objetos, cada qual destinado a desempenhar um papel específico. Além disso, deve criar alguns mistérios e problemas para o jogador resolver.

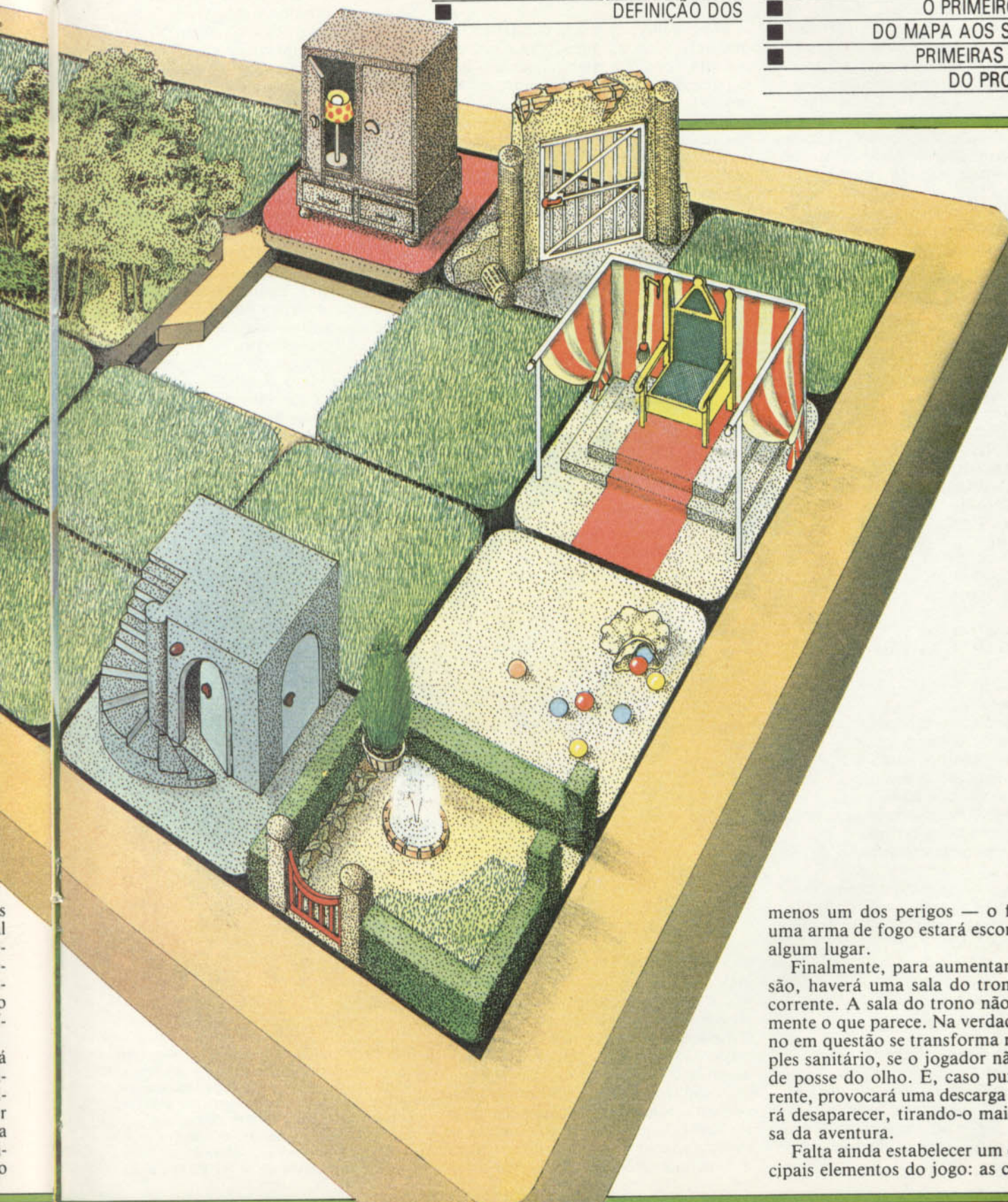
Não é necessário fazer tudo isso de uma vez: à medida que se pensa na história, seus contornos vão se tornando mais nítidos, o que facilita a definição dos detalhes. Comece, portanto, rascunhando o roteiro básico.

Nosso jogador encontra-se em péssima situação financeira e, por isso, sai em busca do fabuloso olho inca (de altíssimo valor), escondido em algum lugar do mundo da aventura. Infelizmente, a Secretaria da Receita enviou um fiscal para acompanhar o caso. O papel desse personagem é bastante semelhante ao do pirata de outras aventuras — ou seja, cabe-lhe arruinar o infeliz jogador. Sua aparição pode acarretar dois acontecimentos. Se o jogador estiver levando consigo um objeto, o fiscal o confiscará para amortizar sua enorme dívida com a Secretaria. Caso o jogador não tenha encontrado nenhum objeto (não podendo, portanto, pagar), o fiscal o prenderá numa masmorra.

Este é o esquema básico da aventura. Resta, agora, trabalhar alguns detalhes — por exemplo, os objetos que serão encontrados na busca. No nosso jogo, decidimos contrariar a regra geral de que todos os objetos devem ser úteis ao desenvolvimento da aventura. Desta vez, haverá um objeto sem nenhuma utilidade na busca empreendida pelo jogador: trata-se de algo pesado (um tijolo, por exemplo), que irá causar a morte de quem tentar atravessar o rio a nado, carregando-o.

O objeto mais importante de todos será o olho. Para aumentar o interesse do jogo, convém imaginar uma maneira de disfarçá-lo ou escondê-lo. Poderíamos colocá-lo dentro de um baú ou de uma catacumba, mas existem soluções mais sutis para enganar o jogador. Assim, em vez de esconder o olho num lugar que obviamente contém algo de valor, vamos deixá-lo dentro de um saquinho de bolas de gude. O aventureiro não irá a lugar nenhum se tentar jogar com as bolinhas de gude!

Um dos artifícios favoritos em jogos de aventura é o quarto escuro, no qual todas as coisas horríveis podem acontecer. Esta aventura não será uma exceção. Sem a lâmpada — que deverá descobrir em algum misterioso lugar — o pobre aventureiro não encontrará as saídas, ficando em péssima situação.

Talvez isso seja um tanto injusto, já que o jogador não receberá nenhum aviso de perigo iminente, nem terá condições de sair do quarto escuro, a não ser que tenha encontrado a lâmpada. Para dar uma melhor oportunidade ao aventureiro, permitindo-lhe enfrentar pelo

menos um dos perigos — o fiscal —, uma arma de fogo estará escondida em algum lugar.

Finalmente, para aumentar a diversão, haverá uma sala do trono e uma corrente. A sala do trono não é exatamente o que parece. Na verdade, o trono em questão se transforma num simples sanitário, se o jogador não estiver de posse do olho. E, caso puxe a corrente, provocará uma descarga que o fará desaparecer, tirando-o mais depressa da aventura.

Falta ainda estabelecer um dos principais elementos do jogo: as condições

necessárias à vitória. Não existe saída evidente do mundo da aventura e boa parte do quebra-cabeças consiste em descobrir como escapar com a peça do totem. Assim, para ganhar o jogo, o aventureiro obviamente precisará ter encontrado o olho — não apenas o saquinho de bolas de gude. Para dificultar um pouco mais, ele deverá também estar na sala do trono. Puxar a corrente desta vez não fará com que ele entre pelo cano!

Transformar a saída em risco, sob condições diferentes, apresenta a vantagem de desencorajar o jogador a tentá-la constantemente; desta maneira prolonga-se o jogo todo.

A AVENTURA ATÉ AQUI

Antes de começar o mapeamento, convém fazer uma recapitulação, listando os elementos até agora envolvidos na aventura. Assim, não se corre o risco de perder o fio da meada.

PERSONAGENS
- Aventureiro
- Fiscal da Receita — deve surgir aleatoriamente

OBJETOS
- Globo ocular — escondido no saquinho de bolas de gude.
- Tijolo — peso que mata o aventureiro caso este tente atravessar o rio a nado
- Lâmpada — necessária para se achar a saída do quarto escuro
- Arma — para matar o fiscal da Receita
- Corrente — tira o aventureiro do jogo se este chegar à sala do trono e puxá-la, sem estar de posse do olho

LOCAIS
- Rio
- Quarto escuro
- Sala do trono

Até aqui, fixamos apenas três dos locais da aventura, em função do que neles deve acontecer. Podemos muito bem, neste ponto, tomar novas decisões. Mas, seja como for, o próximo passo consistirá em encaixar todos os elementos num mesmo mapa do mundo da aventura.

COMO FAZER O MAPA

O primeiro mapa será composto por uma série de quadrados ligados entre si por setas, como mostra a ilustração desta página. Cada quadrado representará um ambiente ou local da aventura — local talvez seja o termo mais adequado, por sua abrangência. Pode designar desde a cabeça de um alfinete escondido na barra do vestido da rainha, até uma imensa planície. Todos os locais devem ser incluídos no mapa: os da lista preliminar e outros que se façam necessários para completar o jogo.

Ao esboçar o mapa, não se esqueça de assinalar a direção em que se pode ir a partir de cada local. Se quiser, inclua saídas que só funcionem em uma direção — acompanhadas de mensagens, como por exemplo:

A PORTA BATE,
FECHANDO-SE ATRÁS DE VOCÊ

As faixas listradas que saem do quarto escuro indicam que o jogador só poderá ir na direção indicada quando determinadas condições forem satisfeitas. No nosso caso, a condição é estar de posse da lâmpada acesa, para enxergar as saídas.

Não é fácil prever o número de locais que determinada quantidade de memória RAM pode conter. A dificuldade está no próprio programa de aventura, que inclui muitos elementos, todos precisando de memória — descrições de locais, palavras que o programa deve reconhecer, o número de objetos e o papel de cada um deles, o número de enigmas e sua complexidade, etc.

Depois de ter programado algumas pequenas aventuras e verificado o espaço que ocupam, ficará mais fácil calcular o que é possível encaixar na memória do seu computador.

Quem tiver um micro de 16K logo descobrirá que é impossível escrever uma aventura em grande escala dispondo de tão pouca memória RAM. No entanto, a aventura que desenvolveremos aqui inclui um número reduzido de locais — doze, ao todo — e, portanto, não dará esse tipo de dor de cabeça.

Um mapa para a Busca do Olho Inca pode ser semelhante ao de nossa ilustração. As ligações entre os locais foram todas previstas, e o ponto de partida estabelecido. Isso é muito importante, pois afeta a maneira de se abordar a aventura, a ordem em que os objetos são descobertos e também a tentativa de

O primeiro mapa da aventura mostra todos os locais planejados em suas posições relativas. As setas vermelhas indicam as saídas que estão constantemente abertas; as setas listradas indicam saídas que só podem ser utilizadas sob condições especiais — neste caso, quando o aventureiro estiver de posse da lâmpada.

se resolver o quebra-cabeças.

Os objetos também estão assinalados em seus locais. Os que aparecerão posteriormente — como, por exemplo, o olho inca — devem ser listados à margem do mapa.

DO MAPA AOS SETORES

Uma vez concluído o mapa, os dados podem ser transferidos para os setores.

Em geral, no planejamento de jogos de aventura, os setores são organizados em dois tipos de conjunto: um que tem por base quadrados e outro que tem por base octógonos (veja as ilustrações na página 230). A escolha de qual utilizar dependerá do número de saídas de cada local.

O tipo mais simples de aventura inclui saídas em quatro direções: norte, sul, leste e oeste (como na Busca do Olho). Nesse caso, deve-se transferir os dados para um conjunto de setores quadrados, de modo que se enquadrem nas condições do mapa proposto. A maneira de fazê-lo será detalhadamente explicada mais adiante.

Se a aventura incluir saídas a noroeste, nordeste, sudeste e sudoeste, deve-se utilizar o conjunto de setores octogonais. O emprego desse tipo de conjunto é, porém, muito complicado.

Existe ainda a possibilidade de se incluir, na aventura, deslocamentos para cima ou para baixo. Nesse caso, a melhor solução é a utilização de conjuntos distintos de setores para cada "nível" da aventura.

A Busca do Olho tem como base um conjunto do tipo quadrado — ou seja, permite saídas somente para norte, sul, leste e oeste. A não ser que haja uma real necessidade de outras direções, esse tipo de aventura é bastante satisfatório, permitindo, inclusive, que se introduza uma certa confusão no direcionamento das saídas. Por exemplo, pode-se acrescentar a seguinte frase a uma descrição:

EXISTE UMA ESCADA
DESCENDO PARA O OESTE

e, para descer as escadas, bastará usar a resposta normal OESTE.

OS SETORES

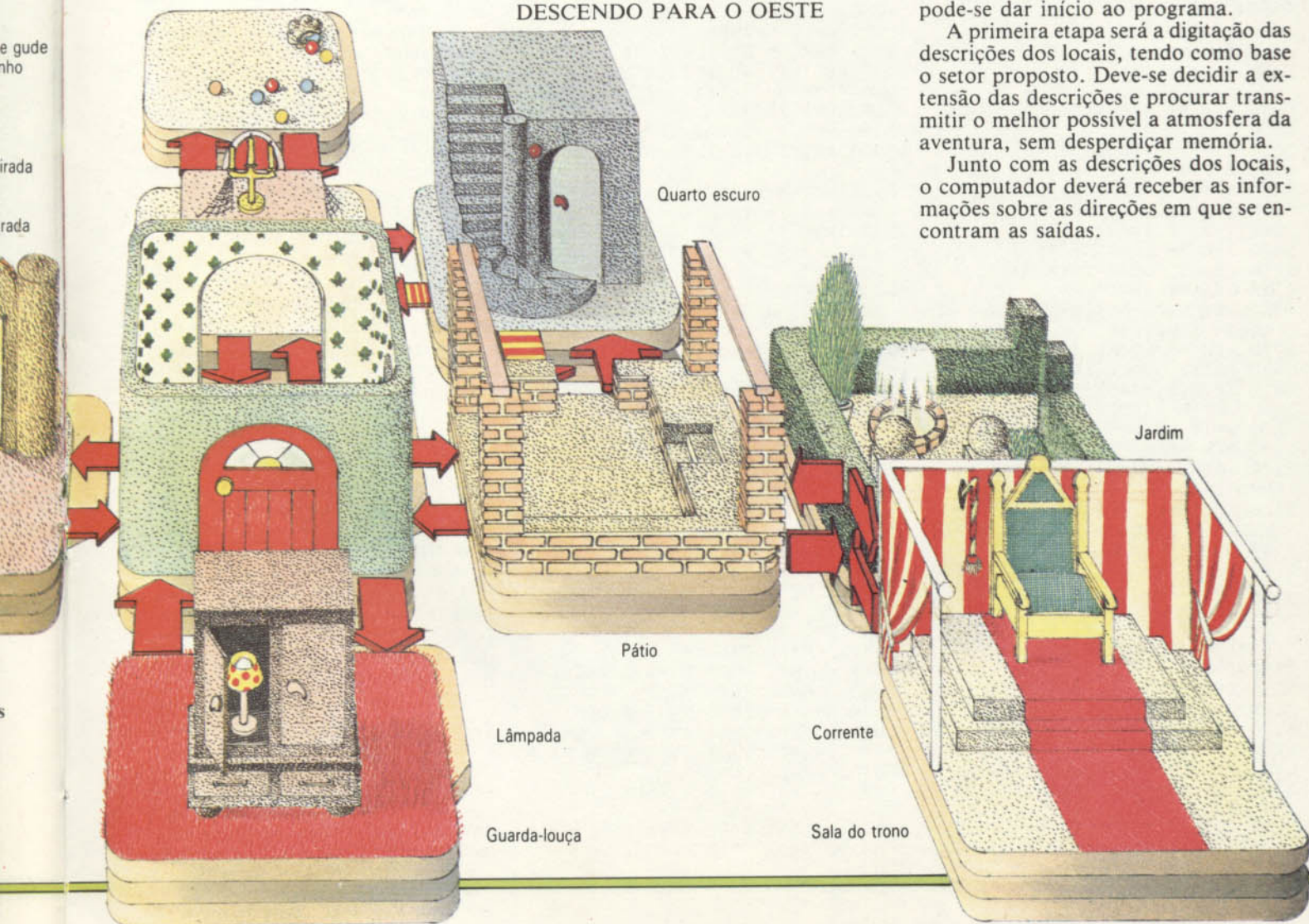
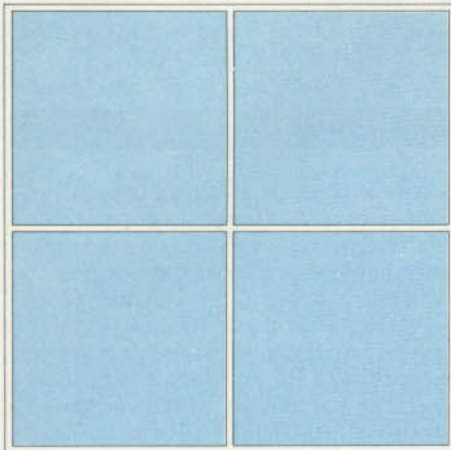
Nossa aventura utilizará um conjunto de setores composto de seis por quatro quadrados — verifique o mapa proposto, para cima e para baixo, para a esquerda e para a direita. Antes de iniciar a transferência de todos os detalhes para os setores, certifique-se de que cada quadrado foi devidamente numerado. Deve-se começar com o número 1, no alto à esquerda, e prosseguir até a parte inferior à direita.

Uma vez numerados os quadrados e transferidos os detalhes, o conjunto dos setores terá uma aparência semelhante à da ilustração da página 231.

COMO COMEÇAR O PROGRAMA

Depois de estruturar o roteiro básico da história e de completar os setores, pode-se dar início ao programa.

A primeira etapa será a digitação das descrições dos locais, tendo como base o setor proposto. Deve-se decidir a extensão das descrições e procurar transmitir o melhor possível a atmosfera da aventura, sem desperdiçar memória.

Junto com as descrições dos locais, o computador deverá receber as informações sobre as direções em que se encontram as saídas.

Parte de um conjunto de setores quadrados. Neste tipo de conjunto, pode-se planejar aventuras utilizando quatro saídas: para norte, sul, leste e oeste.

Parte de um conjunto de setores octogonais — utilizado quando se quer incluir saídas para noroeste, nordeste, sudoeste e sudeste.

Veja, abaixo, as primeiras seções do programa. O número alto das linhas tem a finalidade de assegurar espaço para as seções anteriores, à medida que o jogo for se desenvolvendo. Digite a seção e preserve-a em fita:

```
5000 REM ** DESCRICAO DO LOCAL
**
5010 REM  ** LOCAL 4 **
5020 PRINT "VOCE ESTA DO LADO D
E FORA DE UM GRANDE PREDIO"
5030 LET N=0: LET E=0: LET S=1:
 LET W=0: RETURN
5040 REM ** LOCAL 7 **
5050 PRINT "VOCE ESTA A BEIRA D
E UM GRANDE  RIO"
5060 LET N=0: LET E=1: LET S=0:
 LET W=0: RETURN
5070 REM ** LOCAL 8 **
5080 PRINT "VOCE ESTA NUMA FLOR
ESTA PETRIFI-CADA"
5090 LET N=0: LET E=0: LET S=1:
 LET W=1: RETURN
5100 REM ** LOCAL 10 **
5110 PRINT "VOCE ESTA NUMA SALA
 EMPOEIRADA"
5120 LET N=1: LET E=1: LET S=1:
 LET W=0: RETURN
5130 REM ** LOCAL 11 **
5140 PRINT "VOCE ESTA NUMA SALA
 ESCURA"
5150 IF LA<>1 THEN  LET N=0: LE
T E=0: LET S=0: LET W=0: PRINT
" ESTA MUITO ESCURO PARA VER AS
   SAIDAS": LET DA=1: RETURN
5160 LET N=0: LET E=0: LET S=1:
 LET W=1: RETURN
5170 REM ** LOCAL 14 **
5180 PRINT "VOCE ESTA EM UM ATA
LHO ENLAMEADO"
5190 LET N=1: LET E=1: LET S=0:
 LET W=0: RETURN
5200 REM ** LOCAL 15 **
5210 PRINT "VOCE ESTA NA ENTRAD
A DA CIDADE OCULTA"
5220 LET N=0: LET E=1: LET S=0:
 LET W=1: RETURN
5230 REM ** LOCAL 16 **
5240 PRINT "VOCE ESTA NO HALL D
E ENTRADA"
5250 LET N=1: LET E=1: LET S=1:
 LET W=1: RETURN
5260 REM ** LOCAL 17 **
5270 PRINT "VOCE ESTA NO PATIO"
5280 LET N=1: LET E=1: LET S=0:
 LET W=1: RETURN
5290 REM ** LOCAL 18 **
5300 PRINT "VOCE ESTA NO JARDIM
"
5310 LET N=0: LET E=0: LET S=1:
 LET W=1: RETURN
5320 REM ** LOCAL 22 **
5330 PRINT "VOCE ESTA NO GUARDA
-LOUCAS"
5340 LET N=1: LET E=0: LET S=0:
 LET W=0: RETURN
5350 REM ** LOCAL 24 **
5360 PRINT "VOCE ESTA NA SALA D
O TRONO"
5370 LET N=1: LET E=0: LET S=0:
 LET W=0: RETURN
```

```
5000 REM**DESCRICAO DOS LOCAIS*
*
5010 REM**LOCAL 4**
5020 PRINT" VOCE ESTA DO LADO D
E FORA DE    UMA GRANDE CONSTRU
CAO"
5030 N=0:E=0:S=1:W=0:RETURN
5040 REM**LOCAL 7**
5050 PRINT" VOCE ESTA A BEIRA D
E UM GRANDE  RIO"
5060 N=0:E=1:S=0:W=0:RETURN
5070 REM**LOCAL 8**
5080 PRINT" VOCE ESTA NUMA FLOR
ESTA PETRIFI CADA"
5090 N=0:E=0:S=1:W=1:RETURN
5100 REM**LOCAL 10**
5110 PRINT" VOCE ESTA NUMA SALA
 MUITO SUJA"
5120 N=1:E=1:S=1:W=0:RETURN
5130 REM**LOCAL 11**
5140 PRINT" VOCE ESTA NUM QUART
O ESCURO"
5150 IF OB(6)<>-1 OR LA<>1 THEN
 N=0:E=0:S=0:W=0:PRINT" ESTA MU
ITO ESCURO PARA VER AS   SAIDAS
":RETURN
5160 N=0:E=0:S=1:W=1:RETURN
5170 REM**LOCAL 14**
5180 PRINT" .VOCE ESTA NUM ATALH
O ENLAMEADO"
5190 N=1:E=1:S=0:W=0:RETURN
5200 REM **LOCAL 15**
5210 PRINT" VOCE ESTA NA ENTRAD
A DA CIDADE OCULTA"
5220 N=0:E=1:S=0:W=1:RETURN
5230 REM**LOCAL 16**
5240 PRINT" VOCE ESTA NO HALL D
E ENTRADA"
5250 N=1:E=1:S=1:W=1:RETURN
5260 REM**LOCAL 17**
5270 PRINT" VOCE ESTA NO PATIO"
5280 N=1:E=1:S=0:W=1:RETURN
5290 REM**LOCAL 18**
5300 PRINT" VOCE ESTA NO JARDIM
"
5310 N=0:E=0:S=1:W=1:RETURN
5320 REM**LOCAL 22*
5330 PRINT" VOCE ESTA NO GUARDA
-LOUCAS"
5340 N=1:E=0:S=0:W=0:RETURN
5350 REM**LOCAL 24**
5360 PRINT" VOCE ESTA NA SALA D
O TRONO"
5370 N=1:E=0:S=0:W=0:RETURN
```

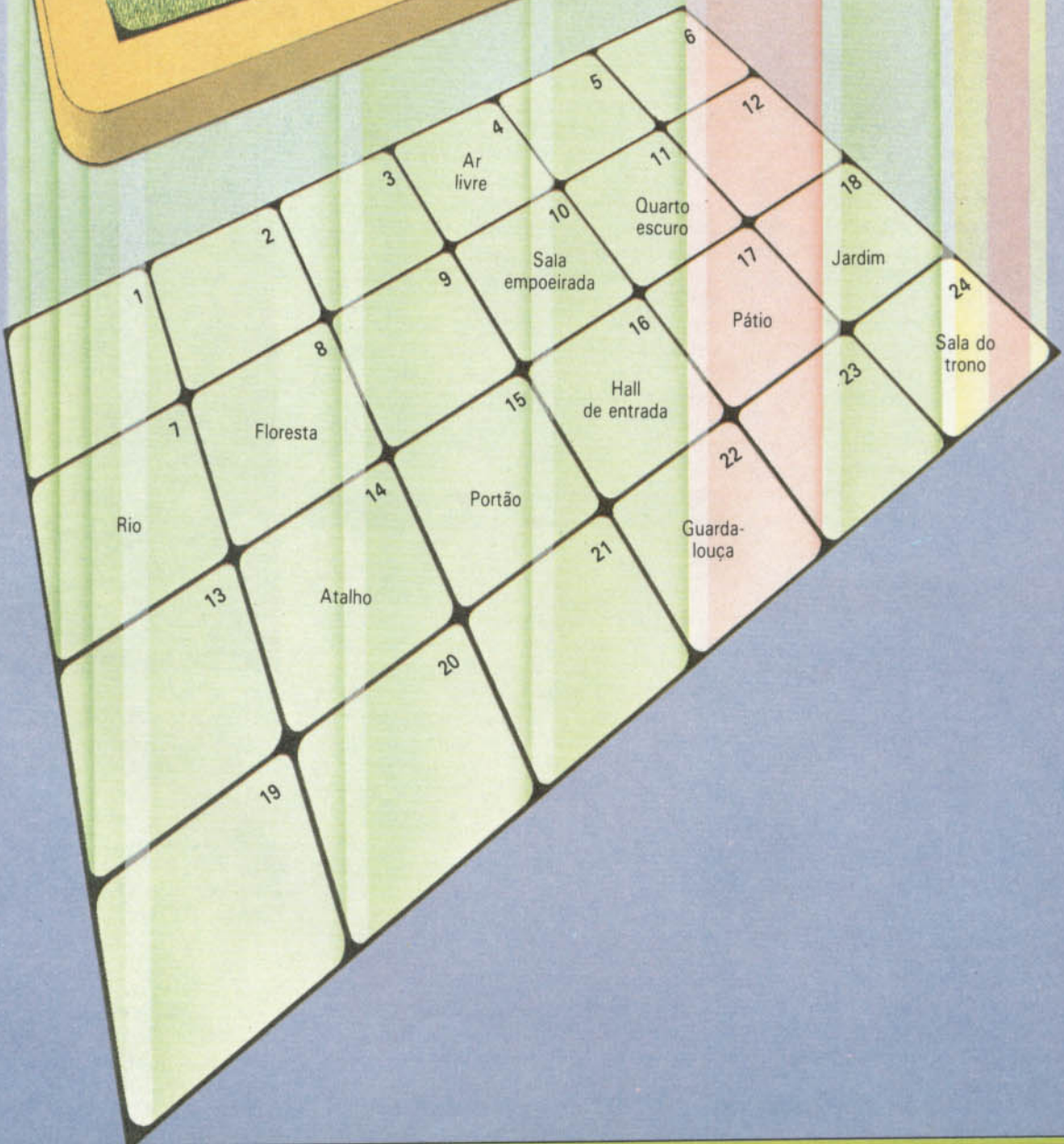
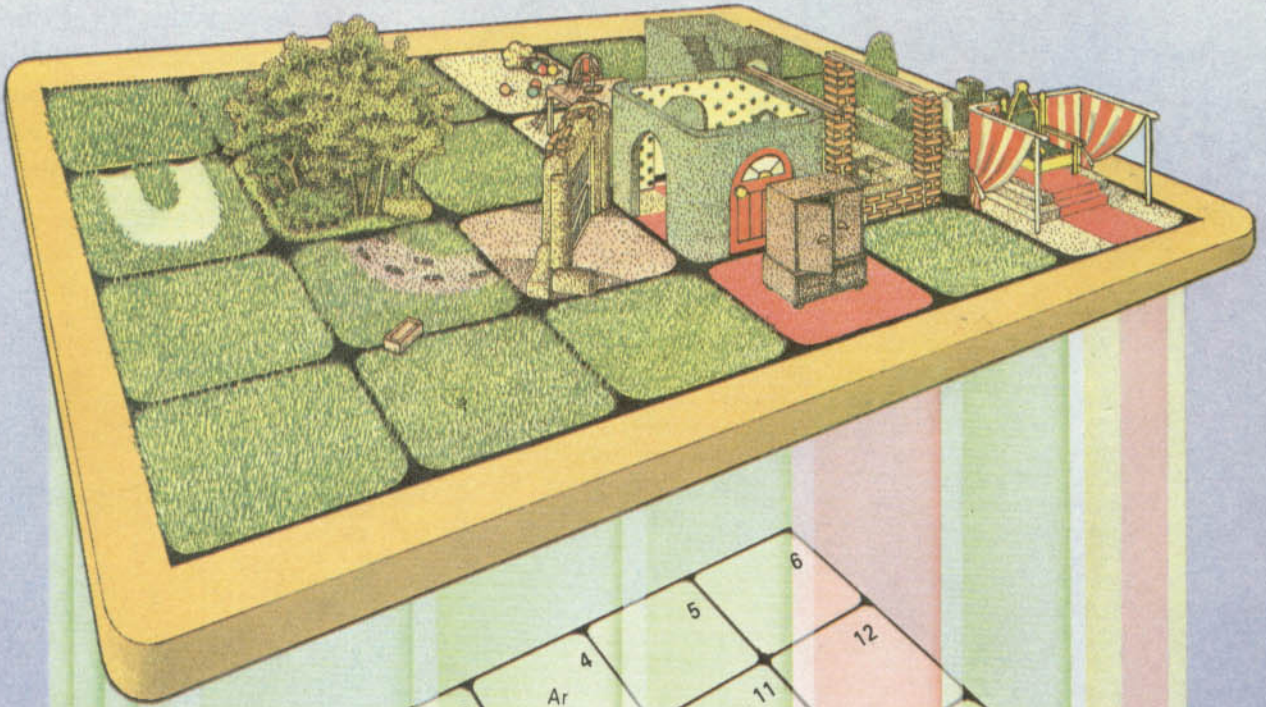
Não se preocupe com o uso repetido de comentários REM incluídos na memória. Nesta etapa inicial do desenvolvimento do programa, o importante é saber o que cada trecho do programa faz, ou qual o número do local a que determinada descrição se refere. Os comentários sempre podem ser eliminados posteriormente. Após cada linha de descrição do local, existe outra linha contendo dados sobre suas possíveis saídas. As variáveis N, S, L e O referem-se a norte, sul, leste e oeste. Elas podem ter um ou dois valores — 0 significa que não há saída naquela direção, ao passo que 1 significa que há uma saída.

Finalmente, existe um **RETURN** após as seções do programa, pois cada descrição de local será chamada por uma instrução **GOSUB**.

Um **IF...THEN** extra na seção do quarto escuro verifica se o aventureiro possui a lâmpada; porém, a descrição das variáveis será tratada mais adiante, quando analisarmos os objetos.

No próximo artigo veremos como movimentar o jogador pelo mundo que criamos.

Última etapa antes da programação. O conjunto de setores é uma cópia direta do mapa, numa forma que facilita o trabalho.

1
2
3
4
Ar livre
5
6
7
Rio
8
Floresta
9
10
Sala empoeirada
11
Quarto escuro
12
13
14
Atalho
15
Portão
16
Hall de entrada
17
Pátio
18
Jardim
19
20
21
22
Guarda-louça
23
24
Sala do trono

RECURSOS GRÁFICOS SOFISTICADOS

Os computadores domésticos oferecem vários recursos ao artista amador. Aprenda novas formas de utilizar os comandos gráficos em BASIC e crie livremente suas ilustrações.

Tendo já dominado os princípios básicos do desenho na tela, você pode começar a empregar alguns recursos gráficos especiais, disponíveis no seu computador. Comandos como **MOVE**, **PLOT**, **DRAW**, **PAINT** e **CIRCLE** permitem que você solte as rédeas de sua imaginação, criando qualquer tipo de imagem — de um diagrama explicativo para ilustrar uma mensagem comercial ao cenário de um excitante jogo de aventuras.

No artigo da página 113 mostramos como utilizar comandos simples para criar ilustrações com linhas na tela e, em alguns casos, como acrescentar cor. Mas você pode sofisticar seu repertório, aprendendo outras maneiras de traçar pontos, desenhar linhas, triângulos, quadrados e círculos. Os comandos aqui ensinados — em combinação com cores ou não — constituem um eficiente recurso para formar imagens estáticas ou até mesmo móveis.

Embora os vários modelos de computador utilizem esses comandos de maneira diferente, todos são capazes de produzir alguns efeitos interessantes. As exceções são o Apple, o TK-2000, o TRS-80 e o ZX-81, cujo BASIC não inclui esses comandos. Entretanto, você pode obtê-los acrescentando um cartucho **ROM** adequado ou adquirindo programas em disco ou fita.

ARCOS E CÍRCULOS

Os arcos e círculos são um dos mais úteis ''instrumentos'' do Spectrum para fazer desenhos na tela.

Este programa para um campo de golfe mostra como empregá-los para desenhar árvores, cercas, água, terrenos acidentados e buracos.

Você pode testar seu progresso na medida em que for rodando cada grupo de linhas. Não dê o comando **NEW** toda hora; se você deixar as linhas intactas, obterá a cena mostrada na ilustração da página 237.

Antes de começar o trabalho com os círculos, porém, convém deixar pronta a sede do clube. Para isso, entre estas linhas no computador:

```
 90 BORDER 4: PAPER 4: CLS
200 LET w=10: LET s=50
210 FOR c=162 TO 174
220 PLOT  INK 2;w,c
230 DRAW  INK 2;s,0
240 LET w=w+2: LET s=s-4
250 NEXT c
260 FOR b=148 TO 162
270 PLOT  INK 2;10,b
280 DRAW  INK 2;50,0
290 NEXT b
295 DRAW  INK 2;10,-3: DRAW 0,
-11
300 PRINT  INK 0;AT 2,2;" ";AT
2,4;" ";AT 2,6;" "
```

As linhas numeradas de 200 a 250 desenham o telhado, utilizando técnicas semelhantes às ensinadas no artigo anterior (página 113). Na tela, em **10.162**, aparece primeiro uma linha de 50 pixels de largura. Depois uma largura de 4 pixels, para cada pixel desenhado.

Um laço parecido aos das linhas 260 a 290 desenha as paredes, com duas linhas extras (na linha 295) para traçar a varanda. Em seguida, a linha 300 preenche as janelas pelo método mais simples possível — imprimindo um quadrado negro, com caracteres gráficos da **ROM**, em três lugares da parede.

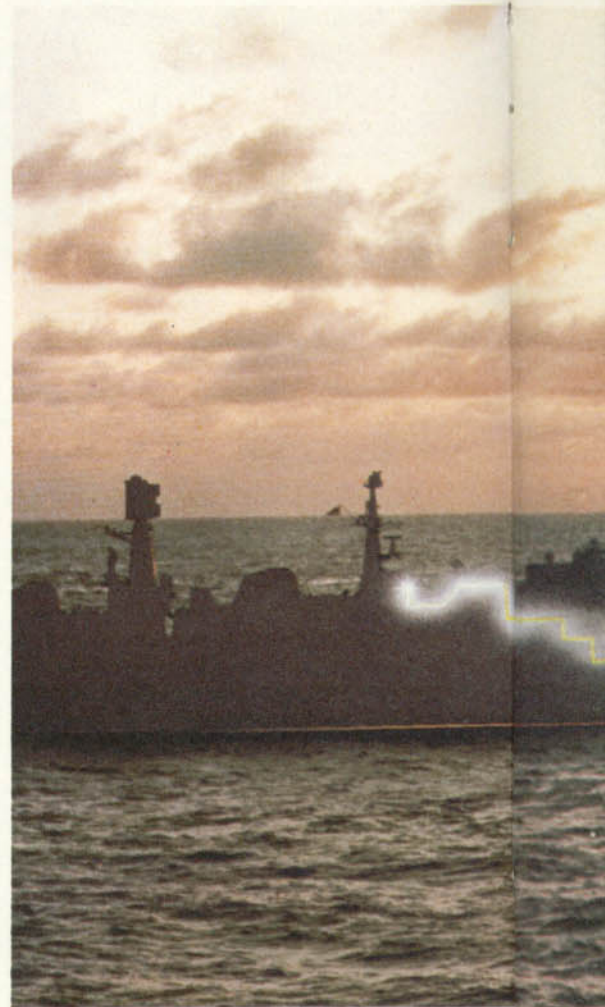
COMO DESENHAR ARCOS

No Spectrum, como já foi explicado anteriormente (página 115), o comando **CIRCLE** é o instrumento mais simples para se desenhar um círculo completo.

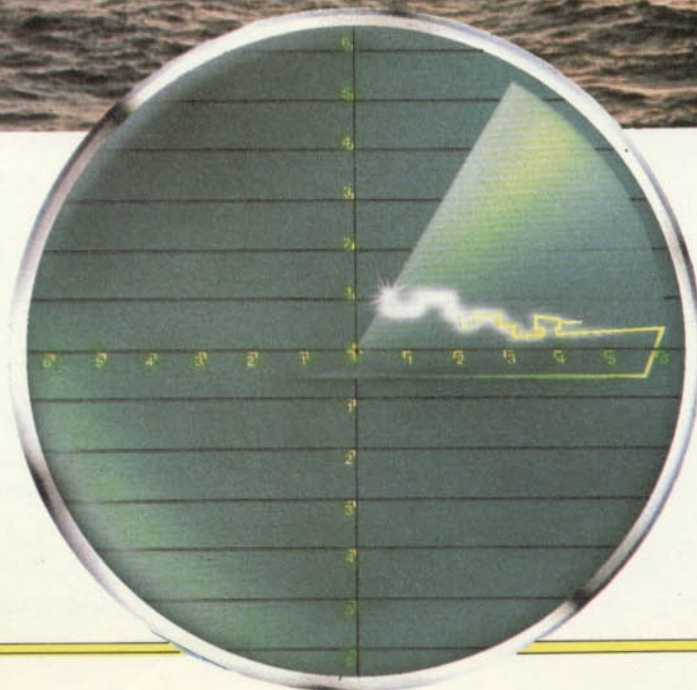
Mas, se quiser somente uma parte do círculo, será mais fácil acrescentar um número a uma declaração convencional **DRAW**. Pode-se obter uma variedade imensa de efeitos por meio dessa declaração; assim, vale a pena fazer alguns experimentos antes de prosseguir.

```
10 PLOT 130,30
20 DRAW 0,10,1
30 GOTO 20
```

(Não se preocupe com a mensagem de erro.)

Como você deve se lembrar, os primeiros dois números da linha 20 dizem ao computador para desenhar uma linha do ponto traçado até um ponto 10 pixels acima dele. O último número, por sua vez, determina a curvatura da linha.

Informa-se assim, ao computador, qual fração do círculo será desenhada. Um círculo completo é representado por duas vezes Pi (3.1416); portanto, com o número 1, obtém-se o desenho de aproximadamente um sexto do círculo (ou, mais precisamente, 1 dividido por duas vezes Pi). Se você tentar modificar a linha 20 para:

```
20 DRAW 0,10,2
```

... obterá uma série de curvas mais pronunciadas. Do mesmo modo, 0,10,3 fornece um modelo denteado, e 0,10,4 produz parte de uma cerca (ou a metade de um tronco de palmeira, dependendo de

sua imaginação), enquanto 0,10,6 produz uma espiral. Se, no entanto, você tentar:

```
20 DRAW 0,10,2*PI
```

... poderá ter uma surpresa! O Spectrum desenhará um círculo em que dois pontos estarão em linha reta. Mas, já que tal círculo pode se tornar muito maior do que sua sala — na verdade, bem maior do que o sistema solar —, o computador desenhará apenas a parte do círculo que é capaz.

Para o cenário, experimente isto:

```
10 PLOT 0,100
20 DRAW RND*5+5,0,2
30 GOTO 20
```

A forma obtida assemelha-se à das ondas em um mar agitado. Para desenhar águas calmas, tente **RND*10+10** ou mesmo **RND*15+10**.

Com relação aos círculos e arcos, vale ainda lembrar que um número negativo no final da linha **DRAW** produzirá a imagem-espelho do arco.

Agora, limpe as linhas de 10 a 30 e continue montando o cenário para a pista de golfe.

A CERCA E O LAGO

O programa da pista de golfe inclui dois exemplos de arcos utilizados para efeito gráfico — o primeiro compõe a pequena cerca em frente à sede do clube; o segundo, o lago.

Digite estas linhas:

```
310 FOR f=0 TO 84 STEP 3
320 PLOT f,142
330 DRAW 3,0,-3
340 NEXT f
350 DRAW 35,-42: DRAW -12,-6
```

O ponto de partida na tela é **0,142**. A linha 330 desenha uma série de arcos estreitos — ou seja, semicírculos com apenas 3 pixels de largura — para formar a cerca. O número obtido é regulado pelo laço **FOR...NEXT**.

Agora, entre e rode estas linhas:

```
100 LET x=130: LET y=125: LET
z=50
110 PLOT  INK 5;x,0
120 DRAW  INK 5;y,z,-1.25
130 LET x=x+1: LET y=y-1: LET
z=z-1
```

```
140 IF x>254 THEN  GOTO 170
150 IF z<1 THEN  LET z=0
160 GOTO 110
```

Aqui a ilustração é mais complicada. Em primeiro lugar, o computador traça um ponto de 130 pixels a partir da esquerda e 0 pixel a partir do canto inferior da tela. Em seguida, desenha uma linha no ponto 125 pixels à direita e a eleva a 50 pixels do canto inferior da tela, "arqueando" a linha em −1.25 radiano, à medida que ela vai se desenvolvendo.

A partir desse estágio as variáveis entram em cena. A variável **x** inicia cada linha 1 pixel mais à direita; a variável **y** torna a linha 1 pixel mais curta do que a anterior (caso contrário ela ultrapassaria os limites da tela), enquanto a variável **z** faz com que a linha termine 1 pixel abaixo da anterior.

Eventualmente, é claro, a variável **z** poderá se tornar um número negativo, fazendo com que o computador tente (mas falhe) imprimir o canto inferior da tela. Por isso a linha 150 é necessária: ela faz com que todas as pequenas linhas no final do programa terminem na linha inferior da tela.

AS ÁRVORES E OS ARBUSTOS

O programa da pista de golfe também utiliza círculos completos (como um substituto para **PAINT** ou uma declaração similar, disponível em alguns computadores) para produzir árvores e arbustos. Estas poucas linhas desenharão alguns arbustos ao acaso no "gramado":

```
400 FOR r=172 TO 168 STEP -1
410 LET x=RND*45+195
415 PLOT x,r-2: DRAW 0,-2
420 CIRCLE x,r,RND*2+1
440 CIRCLE x+10,r,RND*2+1
450 NEXT r
```

Estas outras linhas produzirão algo parecido, no lado direito:

```
460 FOR r=135 TO 172 STEP 6
470 LET y=252
480 CIRCLE y,r,RND+2
490 NEXT r
```

As árvores, no lado inferior esquerdo, são muito grandes e, por isso, não se pode desenhá-las aleatoriamente, usando **DRAW**. Assim, utilizamos uma técnica diferente: **READ...DATA** (veja páginas 128 a 133):

```
900 FOR w=1 TO 3
910 READ a,b
920 PLOT a,b
930 DRAW 0,-24
940 LET f=RND*5+5
950 CIRCLE a-10,b+f,f: CIRCLE
a,b+f,f: CIRCLE a+10,b+f,f
960 CIRCLE a-5,b+f*2,f:
CIRCLE a+5,b+f*2,f
970 CIRCLE a,b+f*3,f
980 NEXT w
3000 DATA 20,70,52,85,84,100
```

A solução, aqui, foi desenhar primeiro os troncos, de modo descendente, a partir dos pontos traçados originalmente. Desse modo, evita-se que eles apareçam através das "folhagens". Os troncos são traçados na tela a **20,70** e assim por diante, pelos dados fornecidos pela linha 3000. A linha 940 desenha aleatoriamente o tamanho das folhagens, enquanto que **b + f**, na linha 950, assegura que as séries de círculos ao fundo iniciem com uma distância razoável acima dos troncos.

TOQUES FINAIS

O **tee** propulsor, ou seja, o apoio onde se coloca a bola para a tacada inicial, é desenhado por estas linhas:

```
1000 LET t=30
1010 FOR y=0 TO 10
1020 PLOT t,y
1030 DRAW -30,30
1040 LET t=t+2
1050 NEXT y
```

E as linhas abaixo desenharão as bandeirinhas no gramado:

```
170 PLOT 220,140
180 DRAW 0,15: DRAW 8,-3
DRAW -8,-2
190 PLOT 22,120
195 DRAW 0,18: DRAW 9,-3:
DRAW -9,-2
```

Finalmente, você precisará de alguns buracos. Na ausência de uma declaração **PAINT**, a maneira de desenhá-los que oferece os melhores resultados (embora também seja a mais demorada) é começar com uma elipse bem pequena e ir aumentando-a, pixel por pixel, até atingir um tamanho razoável.

Como o traçado de elipses será detalhadamente explicado em um artigo futuro sobre as funções matemáticas do computador, entre e rode estas linhas:

```
1495 LET r=1
1500 FOR x=0 TO 2*PI STEP PI/18
0
1510 PLOT  INK 6;168+r*SIN x,14
7+r*COS x/2.5
1520 PLOT  INK 6;235+r*SIN x,10
6+r*COS x/2.75
1530 PLOT  INK 6;225+r*SIN x,97
+r*COS x/2.5
1540 NEXT x
1550 LET r=r+2
1560 IF r>20 THEN  GOTO 6000
1570 GOTO 1500
```

Alternativamente, você poderá fazer os arbustos de um modo mais simples, porém mais grosseiro, utilizando blocos gráficos.

O COMANDO DRAW

Os comandos **CIRCLE** e **LINE** são muito úteis para desenhos simples de formato regular, requerendo um mínimo esforço do programador. Empregá-los para desenhos mais complicados, porém, tornará seus programas longos e de difícil manuseio. Imagine o tamanho de um programa que desenhasse algo como o navio da página 232, usando um comando **LINE** para cada linha da ilustração...

Para contornar esse problema, devemos utilizar o comando **DRAW**, que permite direcionar a trajetória de uma linha enquanto ela está sendo traçada. Podemos dizer ao computador que leve a linha até uma certa distância à direita, até uma outra distância para cima e assim por diante. Essas instruções são fornecidas dentro de um cordão, o que torna o programa muito mais compacto.

Como exemplo, digite e rode este programa; veja como é possível desenhar um navio com algumas poucas linhas.

```
10 PMODE 4,1
20 PCLS 5
30 SCREEN 1,1
40 DRAW "BM23,96C0"
50 DRAW "R28E2U3L6UR6E2R5F2D3R3
U2R7D2R4U9E2R4F2D5R3U2R4U6E3R5D
8"
60 DRAW "R3U24L3UR8DL4D24RF2R5D
4R4U4R6U9E3R5D10R4U2R4U14RD10R3
U2"
70 DRAW "R4D11R7U3E2R4F2D3RD8R3
U5E2R8D2R6U2R7UE2R6F2D4R4U4E2R5
F2"
80 DRAW "R6DL6D3R24G12L195H4U5"
90 PAINT (127,100),0,0
100 GOTO 100
```

```
30 SCREEN 2
40 DRAW "BM23,96C1"
50 DRAW "R28E2U3L6UR6E2R5F2D3R3
U2R7D2R4U9E2R4F2D5R3U2R4U6E3R5D
8"
60 DRAW "R3U24L3UR8DL4D24RF2R5D
4R4U4R6U9E3R5D10R4U2R4U14RD10R3
U2"
70 DRAW "R4D11R7U3E2R4F2D3RD8R3
U5E2R8D2R6U2R7UE2R6F2D4R4U4E2R5
F2"
80 DRAW "R6DL6D3R24G12L195H4U5"
```

```
90 PAINT (127,100),1,1
100 GOTO 100
```

O programa funciona assim:

As linhas numeradas de 10 a 30 estabelecem as condições iniciais, selecionando uma tela de alta resolução.

Os comandos **DRAW** das linhas 40 a 80 formam o contorno do navio. Este comando opera somente no cordão de instruções contidas entre aspas.

A linha 40 é a mais simples, com um curto cordão de instruções que dizem à máquina onde começar o desempenho, e com que cor. A primeira instrução, **BM**, significa "mover em branco", colocando a "caneta" que fará o desenho na posição onde desejamos iniciá-lo. Se não houvesse a letra **B**, a "caneta" traçaria uma reta, enquanto se dirigisse ao ponto de partida, que no nosso caso é **23,96** (note que as coordenadas não estão entre parênteses, como em outros comandos gráficos). A instrução final do cordão seleciona a cor preta (**Ci** no MSX e **CO** no TRS-Color).

A linha 50 começa a traçar o contorno. Embora pareça confuso, o cordão da instrução **DRAW** é na realidade muito simples, consistindo de uma série de direções e distâncias. As letras controlam a direção da linha e os números, sua distância em pixels. Se não houver um número após uma direção, o movimento será de apenas um pixel.

Existem 8 direções: **U**, para cima (*up*, em inglês); **R**, direita (*right*); **D**, para baixo (*down*); **L**, esquerda (*left*); **E**, para cima à direita (45 graus); **F**, para baixo à direita (135 graus); **G**, para baixo à esquerda (225 graus) e **H**, para cima à esquerda (315 graus).

Se acompanharmos os comandos do cordão da linha 50, teremos: à direita 28 pixels, a 45 graus 2 pixels, para cima 3 pixels, e assim por diante. Para se exercitar, tente transformar o cordão em movimentos de lápis, em papel milimetrado.

As linhas 60 a 80 contêm cordões parecidos que completam o contorno do navio. Pode-se utilizar apenas um cordão longo em vez de todas essas linhas, mas um número grande de instruções tende a tornar mais difícil a manipulação e a correção dos dados.

Finalizando o desenho, a linha 90 pinta de preto a silhueta traçada na tela.

COMO DESENHAR LETRAS

Uma das limitações do TRS-Color em relação aos programas gráficos é sua incapacidade de exibir textos na tela de alta resolução (isto não vale para o MSX). Não se pode, por exemplo, imprimir uma contagem de pontos em um jogo que utiliza gráficos (por essa razão, o jogo da página 62 é todo exibido em tela de textos).

Para contornar o problema, você deve projetar seus próprios caracteres utilizando o comando **DRAW**. Digite e rode este programa e você verá a interjeição **ALÔ!** ser traçada:

```
10 PMODE 3,1
20 PCLS
30 SCREEN 1,0
40 OI$="D4R3U4L3BR5D4R3BR2U4R3D
2NL3D2BR2U1BU1U2"
50 DRAW "BM110,50;C3S8"+OI$
60 GOTO 60
```

```
30 SCREEN 2
40 OI$="D4R3U4L3BR5D4R3BR2U4R3D
2NL3D2BR2U1BU1U2"
50 DRAW "BM110,50;C3S8"+OI$
60 GOTO 60
```

No TRS, a mensagem é traçada em **PMODE3**, uma modalidade de quatro cores. No MSX, em **SCREEN 2**, a tela de alta resolução (lembre-se de que o MSX tem outras maneiras de escrever textos nesta tela).

A linha 40 mostra mais uma característica do comando **DRAW**. Como ele opera em um cordão de instruções, você poderá definir variáveis e chamá-las mais tarde com um comando **DRAW** no programa.

As instruções em **OI$** dizem ao computador como traçar as letras para formar **ALÔ!**. Tente transformá-las em movimentos de lápis, como fez anteriormente, lembrando-se de que **B** significa espaço em branco; assim, nenhuma linha aparecerá no desenho quando uma instrução for precedida de **B**.

Se você quiser exibir em telas de gráficos outras palavras ou expressões — **QUE AZAR!** ou **BEM FEITO!**, por exemplo — poderá estabelecer mais cordões, tais como **QA$** e **BF$**. Tente elaborar as instruções para uma dessas expressões em papel gráfico. Uma vez que tenha definido os cordões, você poderá chamá-los de volta sempre que forem necessários ao programa.

O **ALÔ!** é traçado pela linha 50. A posição inicial é **110,50** e a cor é de número 3; a palavra será impressa em tamanho 8 (escala dupla). O **S** pode ser seguido por um número de 1 a 62, no TRS, e de 1 a 255, no MSX. O 1 é um quarto do tamanho, o 4 é o tamanho normal (a escala que se obtém se não houver instrução) e o 8 é o tamanho duplo.

Como demonstra a linha 50, é possível unir cordões de instruções **DRAW** como quaisquer outros cordões. Se você quiser chamar cordões como **QA$** ou **BF$**, poderá fazê-lo do mesmo modo que na linha 50. Movimente para o ponto de partida utilizando **BM** e, então, estabeleça a cor por **C** e o tamanho por **S**.

MENSAGENS MAIS LONGAS

É muito trabalhoso definir cordões para cada mensagem quando se quer exibir na tela um grande número delas.

Para simplificar a tarefa, defina todos os caracteres que irá utilizar, digitando um programa como o que se segue:

```
10 PMODE 3,1
20 DIM LE$(26)
30 PCLS
40 FOR K=0 TO 26
45 READ LE$(K):NEXT
50 FOR K=0 TO 9
55 READ NU$(K):NEXT
60 DATA BR2,ND4R3D2NL3ND2BE2,ND
4R3DGNL2FDNL3BU4BR2,NR3D4R3BU4B
R2,ND4R2FD2GL2BE4BR,NR3D2NR2D2R
3BU4BR2
70 DATA NR3D2NR2D2BE4BR,NR3D4R3
U2LBE2BR,D4BR3U2NL3U2BR2,ND4BR2
,BD4REU3L2R3BR2,D2ND2NF2E2BR2
80 DATA D4R3BU4BR2,ND4FREND4BR2
,ND4F3DU4BR2,NR3D4R3U4BR2,ND4R3
D2NL3BE2,NR3D4R3NHU4BR2
90 DATA ND4R3D2L2F2BU4BR2,BD4R3
U2L3U2R3BR2,RND4RBR2,D4R2U4BR2,
D3FEU3BR2,D4EFU4BR2
100 DATA DF2DBL2UE2UBR2,DFND2EU
BR2,R3G3DR3BU4BR2
110 DATA NR2D4R2U4BR2,BDEND4BR2
,R2D2L2D2R2BU4BR2,NR2BD2NR2BD2R
2U4BR2,D2R2D2U4BR2,NR2D2R2D2L2B
E4,D4R2U2L2BE2BR2,R2ND4BR2,NR2D
4R2U2NL2U2BR2,NR2D2R2D2U4BR2
120 SCREEN 1,0
130 A$="TESTANDO0123456789"
140 DRAW "BM60,50C3S8"
150 GOSUB 9000
160 GOTO 160
9000 FOR K=1 TO LEN(A$)
9010 B$=MID$(A$,K,1)
9020 IF B$>="0" AND B$<="9" THE
N DRAW NU$(VAL(B$)):GOTO 9050
9030 IF B$=" " THEN N=0 ELSE N=
ASC(B$)-64
9040 DRAW LE$(N)
9050 NEXT
9060 RETURN
```

```
20 DIM LE$(26)
30 CLS
40 FOR K=0 TO 26
45 READ LE$(K):NEXT
50 FOR K=0 TC 9
55 READ NU$(K):NEXT
60 DATA BR2,ND4R3D2NL3ND2BE2,ND
4R3DGNL2FDNL3BU4BR2,NR3D4R3BU4B
R2,ND4R2FD2GL2BE4BR,NR3D2NR2D2R
3BU4BR2
70 DATA NR3D2NR2D2BE4BR,NR3D4R3
U2LBE2BR,D4BR3U2NL3U2BR2,ND4BR2
,BD4REU3L2R3BR2,D2ND2NF2E2BR2
80 DATA D4R3BU4BR2,ND4FREND4BR2
,ND4F3DU4BR2,NR3D4R3U4BR2,ND4R3
D2NL3BE2,NR3D4R3NHU4BR2
90 DATA ND4R3D2L2F2BU4BR2,BD4R3
U2L3U2R3BR2,RND4RBR2,D4R2U4BR2,
D3FEU3BR2,D4EFU4BR2
100 DATA DF2DBL2UE2UBR2,DFND2EU
BR2,R3G3DR3BU4BR2
110 DATA NR2D4R2U4BR2,BDEND4BR2
,R2D2L2D2R2BU4BR2,NR2BD2NR2BD2R
2U4BR2,D2R2D2U4BR2,NR2D2R2D2L2B
E4,D4R2U2L2BE2BR2,R2ND4BR2,NR2D
4R2U2NL2U2BR2,NR2D2R2D2U4BR2
120 SCREEN 2
130 A$="TESTANDO0123456789"
140 DRAW "BM60,50C1S8"
150 GOSUB 9000
160 GOTO 160
9000 FOR K=1 TO LEN(A$)
9010 B$=MID$(A$,K,1)
9020 IF B$>="0" AND B$<="9" THE
N DRAW NU$(VAL(B$)):GOTO 9050
9030 IF B$=" " THEN N=0 ELSE N=
ASC(B$)-64
9040 DRAW LE$(N)
9050 NEXT
9060 RETURN
```

O conjunto de caracteres — letras de A a Z e algarismos de 0 a 9 — está contido nas linhas **DATA** de 60 a 110. O conteúdo destas é transferido para duas variáveis alfanuméricas — **LE$** e **NU$** — pelas linhas 40 a 55.

Para utilizar esse conjunto é necessário definir a mensagem a ser escrita, assim como a posição inicial, a cor e o tamanho. A linha 130 coloca em **A$** uma mensagem de teste — **TESTANDO0123456789** — que pode ser substituída por qualquer outra de sua criação. A linha 140 estabelece a posição inicial, a cor e o tamanho. Com a mensagem e suas características definidas, podemos dizer à máquina que desenhe a mensagem. A sub-rotina de impressão — que começa na linha 9000 — examina cada caractere em **A$**, um por vez, acha as instruções apropriadas nos conjuntos (veja programação de jogos, na página 101) e, em seguida, desenha o caractere na tela.

Modificando **A$** na linha 130 e, se necessário, a posição inicial na linha 140, podemos desenhar a mensagem que quisermos. Como a parte do programa que traça as letras é uma sub-rotina, teremos quantas mensagens desejarmos, mas devemos colocá-las sempre em **A$**.

O procedimento será muito útil, sobretudo se tivermos uma série de mensagens para exibir em diversos pontos da tela durante um jogo.

Comece seus programas na linha 120, ou seja, acima das linhas **DATA**; lembre-se, porém, de que as linhas **DIM**, **CLEAR** ou **PCLEAR** em geral precisam ficar no início. Quando quiser escrever uma mensagem, estabeleça cordões com as palavras escolhidas apenas no ponto em que elas forem necessárias ao programa. Em seguida, acrescente o **DRAW**, como na linha 140, e chame a sub-rotina de impressão com **GOSUB 9000**.

E FINALMENTE...

Uma vez elaboradas as instruções para desenhar um perfil, podemos posicioná-lo em qualquer direção, ou mesmo de cabeça para baixo. Basta, para isso, incluir uma nova instrução no cordão: **A**, seguida de um valor de 0 a 3.

A0 posiciona o desenho — no nosso exemplo, a casa — a 0 graus, ou seja, na vertical. **A1** desenha a casa deitada sobre seu lado direito — a 90 graus. **A2**, de cabeça para baixo — a 180 graus. **A3**, finalmente, traça a casa deitada sobre seu lado esquerdo — a 270 graus.

O controle da direção em que a imagem é traçada permite obter o mesmo desenho, de quatro maneiras diferentes, a partir de um único conjunto de instruções. Não será necessário, portanto, informar ao computador como desenhar cada versão.

Para ver como esse recurso funciona, digite e rode o programa a seguir.

```
10 PMODE 3,1
20 PCLS
30 SCREEN 1,0
40 S$="NR16E8F4U4R2D6F2D12L6U6L
4D6L6U12"
50 FOR K=1 TO 20
60 D=RND(200)+27:E=RND(140)+27:
C=RND(3)+1:A=RND(4)-1
70 DRAW"BM"+STR$(D)+","+STR$(E)
+"C"+STR$(C)+"A"+STR$(A)+"XS$;"
80 NEXT K
90 GOTO 90
```

```
30 SCREEN 2
40 S$="NR16E8F4U4R2D6F2D12L6U6L
4D6L6U12"
50 FOR K=1 TO 20
60 D=INT(RND(1)*200)+28:E=INT(R
ND(1)*140)+28:C=INT(RND(1)*15)+
1:A=INT(RND(1)*4)
70 DRAW "BM"+STR$(D)+","+STR$(E
)+"C"+STR$(C)+"A"+STR$(A)+"XS$;
"
80 NEXT K
90 GOTO 90
```

Serão desenhadas 20 casas, todas iguais, exceto na cor e na orientação.

O formato das casas é definido na linha 40. Na linha 60, quatro números aleatórios são escolhidos: **D** e **E** são as coordenadas da posição inicial, **C** é a cor, e **A**, a direção da imagem.

A linha 70 desenha a casa, acrescentando ao cordão todos os parâmetros gerados na linha 60. A função **STR$** converte as variáveis numéricas em cordões — na verdade, coloca o valor destas variáveis entre aspas. Por exemplo, se **D** = **2**, então **STR$** = "**2**".

A linha 70 move, então, a "caneta" para uma posição qualquer e, usando uma cor aleatória, desenha, na direção escolhida ao acaso, as instruções contidas em **S$**. O **X** antes de **S$** significa "execute as instruções contidas em **S$**" e permite a adição de um cordão a outro durante o desenho, sem a necessidade de utilizar o sinal "+".

O emprego do **X** apresenta a vantagem de economizar a "memória de cordões" da máquina — que é limitada —, pois permite unir cordões sem criar novos cordões. O uso do "+", ao contrário, cria um novo cordão, que vai ocupar boa parte da memória que estamos querendo economizar.

O Spectrum usa arcos e círculos para desenhar um campo de golfe.

É possível colocar textos na tela gráfica do Apple II?

Infelizmente, os microcomputadores compatíveis com a linha Apple II não permitem misturar texto e gráficos na mesma tela. Essa limitação reduz bastante o campo de ação do iniciante que quer desenvolver jogos ou outros tipos de programas gráficos que precisam de alguma espécie de título ou identificação na tela.

Como os computadores da linha MSX e TRS-Color, os micros da linha Apple II também dispõem de um comando **DRAW**. Porém, se o objetivo deste recurso é semelhante em todos os computadores mencionados, a filosofia de sua utilização é bastante diferente, quando se trata do Apple.

O funcionamento do comando **DRAW** do Apple baseia-se nas *tabelas de forma* (*shape tables*), que residem em uma certa parte da memória e que determinam um verdadeiro "mapa" de movimentação do cursor gráfico, com base em números binários que identificam quais os bits que devem ficar acesos ou apagados na tela. A especificação de tabelas de forma será explicada em um artigo futuro. Basta saber, por enquanto, que letras e números de qualquer tamanho podem ser desenhados na tela, um a um, com o comando **DRAW**.

ASSEMBLER PARA O APPLE

Traduzir mnemônicos Assembly para código de máquina é uma tarefa extremamente cansativa. Deixe o Apple trabalhar por você: ele o fará com rapidez e facilidade.

O Assembler do Apple é bem mais curto do que os programas correspondentes para o Spectrum, o MSX e o TRS-Color. Isto ocorre porque seu microprocessador — o 6502 — tem bem menos instruções que o Z-80 e o 6809, os microprocessadores das outras máquinas.

Este programa não precisará trabalhar com a infinidade de mnemônicos do Z-80, nem terá de calcular os pós-bytes do 6809.

Devido ao seu tamanho, ele é mais rápido — o que não significa, porém, que sua velocidade seja comparável à dos programas escritos em código. Ele levará alguns minutos para traduzir um longo programa em Assembly.

É bom lembrar que o TK-2000 já vem com um míni-Assembler embutido.

O ASSEMBLER

```
10  HOME
20  DIM K$(77),K1(77),K2(77):H$
 = "0123456789ABCDEF":R$ =  CHR
$ (13)
30  DIM T$(200),RR(100),Z$(100)
40 P = 0: FOR II = 1 TO 76: REA
D K$(II),K1(II),K2(II): IF K$(I
I) <  > "*" THEN  NEXT
50  DATA    ADC,105,101,AND,41,
37,ASL,,6,BCC,,144,BCS,,176,BEQ
,,240,BIT,,36,BMI,,48
60  DATA  BRK,,,BYT,,-256
70  DATA  BNE,,208,BPL,,16,BVC
,,80,BVS,,112,CLC,24,,CLD,216,,
CLI,88,,CLV,184,
80  DATA  CMP,201,197,CPX,224,2
28,CPY,192,196,DEC,,198,DEX,202
,,DEY,136,,EOR,73,67
90  DATA  INC,,230,INX,232,,INY
,200,,JMP,,68,JSR,,24
100  DATA ,,,,,,,,,LDA,169,165,
LDX,162,166,LDY
110  DATA  160,164,LSR,,70,,,,,
,,,,,NOP,,234,,,,,,,,,,ORA,9,5,
PHA,72,,PLA,104,,PLP
120  DATA  40,,PHP,8,,,,,,,,ROL
,,38,ROR,,102
130  DATA   RTI,64,,RTS,96,,SBC
,233,229,SEC,56,,SED,248,,SEI,1
20,,STA,,133,STX,,134
140  DATA  STY,,132,TAX,170,,TA
Y,168,,TSX,186,,TXA,138,,TXS,15
4,,TYA,152,
150  DATA  WOR,,-256,*,,
160  HOME : HTAB 16: VTAB 1: IN
VERSE : PRINT "MENU": NORMAL :
PRINT : PRINT : PRINT
170  PRINT  TAB( 10);"1 - LER O
 DISCO": PRINT : PRINT  TAB( 10
);"2 - GRAVAR NO DISCO": PRINT
: PRINT  TAB( 10);"3 - MONTAR"
180  PRINT : PRINT  TAB( 10);"4
 - EDITAR LIMHA": PRINT : PRINT
  TAB( 10);"5 - APAGAR LINHA":
PRINT : PRINT  TAB( 10);"6 - LI
STAR"
190  PRINT : PRINT  TAB( 10);"7
 - SAIDA": PRINT : PRINT : PRIN
T  TAB( 10);"SELECIONE A OPCAO
";
200  GET A$: IF  VAL (A$) < 1 O
R  VAL (A$) > 7 THEN 200
210 JJ =  VAL (A$): HOME
220  ON JJ GOSUB 1080,1100,250,
1120,1200,1230,1300: PRINT : PR
INT "QUALQUER TECLA PARA CONTIN
UAR"
230  GET A$: IF A$ = "" THEN 23
0
240  GOTO 160
250 K0 = 0:K9 = 0:PS = 0:P0 = 0
260 PS = PS + 1: IF PS <  = 3 T
HEN K = K0:P = P0: PRINT "ETAPA
 ";PS: GOTO 280
270 P0 = P: RETURN
280  GOSUB 980
290  GOSUB 910:OP$ = I$: IF  LE
FT$ (OP$,1) = "*" AND PS = 3 TH
EN  PRINT OP$
300  IF  LEFT$ (OP$,1) = "*" TH
EN 280
310  IF OP$ = "END" AND PS = 3
THEN  PRINT : PRINT " FIM. ENDE
RECO FINAL: ";P - 1
320  IF OP$ = "END" THEN 260
330  IF OP$ <  > "ORG" AND OP$
<  > "P%=" THEN 370
340  GOSUB 910:S = 0: IF  LEFT$
 (I$,1) = "*" THEN S = P:I$ =
RIGHT$ (I$, LEN (I$) - 1)
350 P =  VAL (I$) + S: IF PS =
3 THEN  PRINT "     ORG";P
360  GOTO 280
370  IF P = 0 THEN  PRINT "FALT
A ORG": RETURN
380  IF OP$ <  > "$P%=" AND OP$
 <  > "TXT" THEN 420
390  GOSUB 910: IF PS = 3 THEN
 PRINT "     TXT";I$; TAB( 20);
400  FOR J = 2 TO  LEN (I$):BY
=  ASC ( MID$ (I$ + I$,J,1)): I
F J >  =  LEN (I$) THEN BY = 13
410  GOSUB 870: NEXT J: GOTO 28
0
420  IF  LEN (OP$) <  > 3 THEN
 FOR I = 1 TO 1: GOTO 440
430  FOR I = 1 + 3 * ( ASC (OP$
) - 65) TO 76: IF OP$ = K$(I) T
HEN 490
440  NEXT : IF PS = 3 THEN  PRI
NT OP$
450  IF  LEFT$ (I$,1) = "." THE
N I$ =  RIGHT$ (I$, LEN (I$) -
1)
460  GOSUB 1010:RR(Q2) = P: IF
I$ = "" THEN 290
470  IF PS = 3 THEN  PRINT "(LI
NHA NAO RECONHECIDA)"
480  GOTO 280
490 AD$ = "#0":R = 0: IF K2(I)
<  > 0 THEN  GOSUB 910:AD$ = I$
:OP = K2(I)
500  IF PS = 3 THEN  PRINT " ";
OP$;: IF K2(I) <  > 0 THEN  PRI
NT " ";AD$;
510  IF  LEFT$ (AD$,1) = "#" TH
EN AD$ =  RIGHT$ (AD$, LEN (AD$
) - 1):OP = K1(I)
520  IF  RIGHT$ (AD$,1) <  > ")
" THEN 530
525 OP = OP - 20:AD$ =  MID$ (A
D$,2, LEN (AD$) - 2): IF OP = 4
8 THEN OP = 100
530  IF  RIGHT$ (AD$,2) = ";X"
THEN OP = OP + 16:AD$ =  LEFT$
(AD$, LEN (AD$) - 2)
540  IF AD$ = "A" THEN AD$"0":O
P = OP + 4
550  IF  RIGHT$ (AD$,2) <  > ";
Y" THEN 580
560 OP = OP + 16:AD$ =  LEFT$ (
AD$, LEN (AD$) - 2): IF  RIGHT$
 (OP$,1) <  > "X" THEN OP = OP
- 4:R = 65536
570  IF  RIGHT$ (AD$,1) = ")" T
HEN AD$ =  MID$ (AD$,2, LEN (AD
$) - 2):R = R - 65536
580 S = 1
590  IF AD$ = "" THEN 770
600 X$ =  LEFT$ (AD$,1): IF  LE
N (AD$) = 1 THEN BD$ = "": GOTO
 610
605 BD$ =  RIGHT$ (AD$, LEN (AD
$) - 1)
610  IF X$ = "*" THEN R = R + P
 * S:AD$ = BD$: GOTO 580
620  IF X$ = "+" THEN AD$ = BD$
: GOTO 590
630  IF X$ = "-" THEN AD$ = BD$
:S =  - S: GOTO 590
640 Q = 0
650  IF (X$ <  > "$" AND X$ <
> "&") OR BD$ < "0" OR BD$ >  =
 "G" THEN 690
660  FOR Q2 = 0 TO 15: IF  LEFT
$ (BD$,1) <  >  MID$ (H$,Q2 + 1
,1) THEN 680
670 Q = Q * 16 + Q2: IF  LEN (B
```

- TRADUÇÃO DE MNEMÔNICOS ASSEMBLY PARA CÓDIGO HEXADECIMAL
- CÁLCULO DE SALTOS E DESVIOS
- MANIPULAÇÃO DE RÓTULOS
- TRANSFERÊNCIA DO PROGRAMA EM CÓDIGO PARA A MEMÓRIA

```
D$) = 1 THEN BD$ = "": GOTO 660
675 BD$ = RIGHT$ (BD$, LEN (BD
$) - 1): GOTO 660
680  NEXT :R = R + Q * S:AD$ =
BD$: GOTO 580
690  IF X$ < "A" OR X$ > "Z" TH
EN 720
700 I$ = AD$: GOSUB 1010: IF I$
 < > "" THEN  GOSUB 1050
710 R = R + RR(Q2) * S:AD$ = I$
: GOTO 580
720  IF X$ < "0" OR X$ > "9" TH
EN R = 0: GOTO 760
730  IF AD$ < "0" OR AD$ > "9"
THEN 750
740 Q = Q * 10 + ASC (AD$) - 4
8: IF  LEN (AD$) = 1 THEN AD$ =
 "": GOTO 730
745 AD$ =  RIGHT$ (AD$, LEN (AD
$) - 1): GOTO 730
750 R = R + Q * S: GOTO 580
760  IF PS = 3 THEN  PRINT "(EN
DERECAMENTO INVALIDO)";
770 PP = (OP - ( ABS (OP) > =
8) * 8 *  INT (OP / 8) > = 4)
* 4 + (OP > = - 4 AND OP < =
 - 1) * 4
772 RP = R - P - 2:RP =  ABS (
ABS ((RP < 0) *  INT (RP / 256)
) * 256 -  ABS (RP - (RP > = 2
56) * 256 *  INT (RP / 256)))
775  IF  LEFT$ (OP$,1) = "B" AN
D (0 = PP AND OP > 0) THEN R =
RP
780  IF  LEFT$ (OP$,1) = "J" OR
  LEFT$ (OP$,1) = "W" THEN R =
R + 65536
790  IF R > 255 THEN OP = OP +
8
800 K2 = K2(I):PA =  ABS ( ABS
((OP < 0) *  INT (OP / 16)) * 1
6 -  ABS (OP - (OP > = 16) * 1
6 *  INT (OP / 16)))
805  IF PA = 10 THEN K2 = 0
810  IF PS = 3 THEN  PRINT  TAB
( 16);:BY = P / 256: GOSUB 890:
BY = P - 32768: GOSUB 890: GOSU
B 850
820  IF OP > = 0 THEN BY = OP
/ 256: GOSUB 870:BY = OP: GOSUB
 880
830  IF K2 = 0 THEN 280
840  GOSUB 850:BY = R - 256 *
INT (R / 256): GOSUB 880:BY = R
 / 256: GOSUB 870: GOTO 280
850  IF PS = 3 THEN  PRINT " ";
860  RETURN
870  IF  INT (BY) < = 0 THEN
RETURN
880 P = P + 1:YY =  ABS ( ABS (
(BY < 0) *  INT (BY / 256)) * 2
56 -  ABS (BY - (BY > = 256) *
 256 *  INT (BY / 256))): IF PS
 = 3 THEN  POKE P - 1,YY
890 BY =  ABS ( ABS ((BY < 0) *
  INT (BY / 256)) * 256 -  ABS
(BY - (BY > = 256) * 256 *  IN
T (I / 256)))
895 YY =  ABS ( ABS ((BY < 0) *
  INT (BY / 16)) * 16 -  ABS (B
Y - (BY > = 16) * 16 *  INT (B
Y / 16))): IF PS = 3 THEN  PRIN
T  MID$ (H$,BY / 16 + 1,1); MID
$ (H$,YY + 1,1);
900  RETURN
910  IF K > N THEN I$ = "END":
RETURN
920 K1 = K9 + 1: IF K9 > = LE
N (T$(K)) THEN I$ = "/FALTANDO/
": RETURN
930 K9 = K1: IF  MID$ (T$(K),K1
,1) = " " THEN 920
940  IF K9 >  LEN (T$(K)) THEN
I$ =  MID$ (T$(K),K1,K9 - K1):
RETURN
950  IF  MID$ (T$(K),K9,1) < >
 " " THEN K9 = K9 + 1: GOTO 940
960 I$ =  MID$ (T$(K),K1,K9 - K
```

```
1)
970  RETURN
980  IF K9 <  =  LEN (T$(9)) AN
D PS = 3 THEN  IF  LEN (T$(K))
> K9 - 1 THEN  PRINT  RIGHT$ (T
$(K), LEN (T$(K)) - K9 + 1);
990 K = K + 1:K9 = 0: IF PS = 3
 THEN  PRINT
1000  RETURN
1010 X$ = ""
1020  IF I$ < "A" OR I$ >  = "[
" THEN 1040
1030 X$ = X$ +  LEFT$ (I$,1): I
F  LEN (I$) = 1 THEN I$ = "": G
OTO 1020
1035 I$ =  RIGHT$ (I$, LEN (I$)
 - 1): GOTO 1020
1040  IF I$ <  > "" THEN RETURN
1050  FOR Q2 = 1 TO VV: IF X$ =
 Z$(Q2) THEN 1070
1060  NEXT :VV = VV + 1:Z$(VV)
= X$:Q2 = VV:RR(VV) = 32768
1070  RETURN
1080  INPUT "NOME DO ARQUIVO ";
AR$:D$ = "": REM   CTRL-D
1090  PRINT D$;"OPEN";AR$: PRIN
T D$;"READ";AR$: INPUT N: FOR J
 = 1 TO N: INPUT T$(J): NEXT :
PRINT D$;"CLOSE";AR$: RETURN
1100  INPUT "NOME DO ARQUIVO ";
AR$:D$ = "": REM   CTRL-D
1110  PRINT D$;"OPEN";AR$: PRIN
T D$;"WRITE";AR$: PRINT N: FOR
J = 1 TO N: PRINT T$(J): NEXT :
 PRINT D$;"CLOSE";AR$: RETURN
1120 K = 0: INPUT "QUAL O NUMER
O DA LINHA ";K: HOME
1140 IP$ = "": PRINT K;: INPUT
IP$: IF IP$ = "" THEN  RETURN
1150 K2 = K / 10: IF K2 > N THE
N K2 = N + 1:N = N + 1
1160 IF K2 < .1 THEN K2 = .1
1170 IF K2 =  INT (K2) THEN 11
 90
1180 K2 =  INT (K2) + 1: FOR K3
 = N TO K2 STEP  - 1:T$(K3 + 1)
 = T$(K3): NEXT :N = N + 1
1190 T$(K2) = IP$:K = K + 10: G
OTO 1140
1200 K = 0: INPUT "QUAL O NUMER
O DA LINHA ";K:K2 = K / 10
1210  IF K2 > N OR K2 < 1 OR K2
 >  INT (K2) THEN  RETURN
1220  FOR K3 = K2 TO N:T$(K3) =
 T$(K3 + 1): NEXT :N = N - 1: R
ETURN
1230  IF N = 0 THEN  RETURN
1240 K = 0:K2 = 0: HOME : INPUT
 "QUAL A PRIMEIRA E A ULTIMA LI
NHA ";K,K2:K1 = K / 10:K2 = K2
/ 10
1250  IF K2 > N THEN K2 = N
1260  IF K1 < 1 THEN K1 = 1
1270  HOME : FOR K3 = K1 TO K2:
 PRINT " ";K3 * 10;" ";T$(K3):
NEXT : RETURN
1300  END
```

Nas linhas 1080 e 1100, a variável **D$** deve conter o caractere produzido pelas teclas <**CTRL**> e <**D**> pressionadas ao mesmo tempo. **D$**, portanto, não está vazia. **CTRL + D** é um caractere de controle para uso do disco.

COMO FUNCIONA O PROGRAMA

Uma vez digitado o programa, rodeo. Normalmente, ele é colocado na área livre da memória que vai de 300 até 3EF, em hexa, ou de 768 até 1007, em decimal. Se quisermos colocá-lo em código em outro local, devemos proteger esta área da memória (veja página 88).

Depois que o programa for rodado, surgirá um menu na tela. Para entrar um programa em Assembly, deve-se escolher a opção 4, que diz "EDITAR LINHA". O Assembler usa linhas BASIC com números múltiplos de 10.

A primeira linha — número 10 — deve conter o endereço inicial do programa em código. Ela ficará mais ou menos como esta:

10 org 800

... determinando que o programa em código seja colocado na memória a partir deste endereço. Se tivermos protegido outra área da memória para colocar nosso programa em código, a origem deve ser o endereço inicial daquela porção da memória.

Depois da primeira linha, os números de linha múltiplos de dez aparecem automaticamente, toda vez que se pressiona a tecla **RETURN**. Cada linha deve conter apenas um mnemônico e seus operandos. Caso se pretenda inserir uma linha entre duas já existentes, utiliza-se um número de linha intermediário — e o programa se encarrega da inserção. Os rótulos, ou *labels* — que indicam os locais para onde o programa será desviado —, devem ficar no início da linha, precedidos de um ponto. Os rótulos precisam ter mais de uma letra e não podem ser iguais a nenhum mnemônico. São empregados mnemônicos padrão do 6502, com uma exceção: o uso do ponto e vírgula no lugar da vírgula. Isto é necessário porque o programa utiliza a instrução **INPUT** para receber as linhas. Uma vírgula provocaria a mensagem de "EXTRA IGNORADO".

Caso deseje evitar a tradução de uma linha, coloque um asterisco antes dela. Ao pressionar **RETURN**, surgirá a próxima linha, mas se a mesma tecla for pressionada novamente, sem que qualquer coisa tenha sido escrita na linha, o programa retornará ao menu.

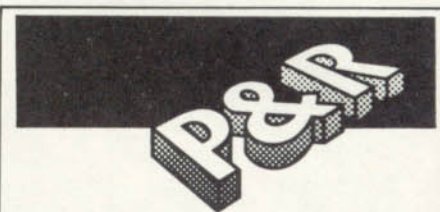
O menu fornece a opção de listar o programa. Se outra tecla for pressionada, volta-se ao menu. Podemos também optar por apagar uma linha, ou simplesmente escrever a nova, com o mesmo número no modo de edição.

Quando estivermos satisfeitos com o programa em Assembly que digitamos, podemos acionar a opção de "montagem" do programa. O Apple mostrará, então, os mnemônicos acompanhados dos códigos hexa correspondentes, ao mesmo tempo em que coloca estes códigos na memória.

Pode-se gravar o Assembler em fita ou disco, como qualquer outro programa (veja página 53). Os mnemônicos — ou programa fonte — devem ser gravados e lidos em disco usando as opções correspondentes.

O programa em código, por sua vez, pode ser gravado em fita ou disco utilizando-se o monitor do Apple, ou por meio do comando **BSAVE**.

Para rodá-lo, deve-se sair do Assembler, por meio da opção "SAÍDA", e digitar:

CALL 800

O que fazer quando um programa longo — como este Assembler — não funciona depois de digitado?

Mesmo o programador mais experiente tem problemas ao digitar programas longos. Sempre se comete algum erro, e é muito difícil localizá-lo.

A maioria dos erros provoca uma mensagem que orienta o programador na direção correta (veja página 141). Mas programas longos seguem caminhos tortuosos, entram em laços e fazem saltos, de modo que o número de linha indicado na mensagem de erro algumas vezes não ajuda em nada.

Por tudo isso, é necessário contar com uma função de rastreamento, como a função **TRACE**, disponível no Apple. Com ela podemos ver o número da linha que está sendo executada, o que torna mais fácil diagnosticar os erros. Para utilizá-la, basta digitar **TRACE** no modo imediato, isto é, sem número de linha na frente. Depois, rode o programa e veja o rastreamento em ação. A função é desativada por **NONTRACE**.

Assim, não descarte o programa se ele não funcionar na primeira tentativa. Procure pelos erros mais comuns. Se ocorrer um "**OUT OF DATA**", por exemplo, verifique com atenção suas linhas **DATA**. Se você tiver esquecido um número, ou mesmo uma vírgula, o programa não funcionará.

CADEIAS DE CARACTERES

- COMPARE E CLASSIFIQUE CORDÕES ALFANUMÉRICOS
- COMO FRACIONAR CORDÕES
- UTILIZE CORDÕES PARA PROCESSAMENTO DE PALAVRAS

Empregadas em quase todos os tipos de programas, as cadeias de caracteres (ou cordões) são muito úteis para quem quer trabalhar com algo mais do que simples números.

Um cordão (ou *string*, em inglês) é, como já vimos, um conjunto de caracteres. Estes, por sua vez, podem ser letras, números, sinais de pontuação ou quaisquer outros símbolos disponíveis no teclado.

Normalmente, um cordão contém uma ou mais informações úteis, ou mesmo partes de informação. Por exemplo, o cordão "PEDRO SILVA 241067 S" engloba diversos dados sobre uma única pessoa: nome, sobrenome, data de nascimento e estado civil. Esses dados são considerados como uma mesma comunicação. A data de nascimento pode ser dividida em dia, mês e ano; assim, na verdade, existem no total seis partes distintas de informação.

Em casos como o do exemplo acima, faz-se necessário fracionar o cordão para extrair as diferentes partes da comunicação. Em outros, ao contrário, pode ser necessário unir dois ou mais cordões. Há casos, ainda, em que se torna imprescindível medir a extensão de um cordão (número total de caracteres) e calcular o valor de algumas de suas partes numéricas. Tudo isso é possível quando se utilizam certas funções e declarações disponíveis no BASIC da maioria dos computadores.

Conhecida como concatenação, a mais simples dessas operações consiste em unir cordões. Para efetuá-la, emprega-se o símbolo "+". Assim, se **A$** for igual a "TUDO" e **B$** igual a "BEM", então **A$** + **B$** será "TUDO BEM". A concatenação junta os cordões sem acrescentar nada a eles. Assim, "439" + "241" é igual a "439241" e não a 680.

COMO COMPARAR CORDÕES

Assim como é capaz de concatenar cordões, o computador pode compará-los para verificar se são iguais, como nesse simples jogo de adivinhação.

Este programa funcionará no ZX-81 se você separar todas as linhas de declarações múltiplas.

```
10 LET G=1: GOTO INT (RND*6)*
10+10
20 LET B$="MACA": GOTO 80
30 LET B$="LARANJA": GOTO 80
40 LET B$="BANANA": GOTO 80
50 LET B$="LIMAO": GOTO 80
60 LET B$="FRAMBOESA": GOTO
80
70 LET B$="ABACAXI": GOTO 80
80 CLS : PRINT "EU SOU UMA FR
UTA. QUE FRUTA SOU EU?"
 90 INPUT A$
100 IF A$=B$ THEN  GOTO 160
110 LET G=G+1
120 PRINT "ERRADO"
130 FOR J=1 TO 200
140 NEXT J
150 GOTO 90
160 IF G=1 THEN  PRINT "VOCE A
CERTOU NA PRIMEIRA TENTATIVA":
STOP
170 PRINT "VOCE ACERTOU APOS "
;G;" TENTATIVAS"
180 STOP
```

TT

```
10 G=1:ON RND(6) GOTO 20,30,40,
50,60,70
20 B$="MACA":GOTO 80
30 B$="LARANJA":GOTO 80
40 B$="BANANA":GOTO 80
50 B$="LIMAO":GOTO 80
60 B$="FRAMBOESA":GOTO 80
70 B$="ABACAXI"
80 CLS:PRINT"EU SOU UMA FRUTA.
 QUE FRUTA SOUEU?"
90 INPUT A$
100 IF A$=B$ THEN GOTO 160
110 G=G+1
120 PRINT"ERRADO!"
130 FOR J=1 TO 1000
140 NEXT J
150 GOTO 90
160 IF G=1 THEN PRINT"VOCE ACER
TOU EM UMA TENTATIVA" ELSE PRIN
T"VOCE ACERTOU APOS";G;"TENTATI
VAS"
```

```
5 R=RND(-TIME)
10 G=1:ON INT(RND(1)*6)+1 GOTO
20,30,40,50,60,70
20 B$="MAÇÃ":GOTO 80
30 B$="LARANJA":GOTO 80
40 B$="BANANA":GOTO 80
50 B$="LIMÃO":GOTO 80
60 B$="MARACUJÁ":GOTO 80
70 B$="ABACAXI":GOTO 80
80 CLS:PRINT"Adivinhe: que frut
a sou eu?"
90 INPUT A$
100 IF A$=B$ THEN 160
110 G=G+1
120 PRINT"Você errou!"
130 FOR J= 1 TO 600
140 NEXT
150 GOTO 90
160 IF G=1 THEN PRINT"Você acer
tou na primeira!" ELSE PRINT"Vo
cê tentou";G;"vezes até acertar
..."
```

Copie o programa anterior sem a linha 5 e troque **CLS** por **HOME**. Modifique a linha 160 e adicione a 170 como se segue:

```
160 IF G=1 THEN PRINT"Voce acer
tou na primeira!":END
170 PRINT"Voce tentou ";G;" vez
es..."
```

Aqui, a linha 10 coloca o contador de palpites em 1 e "lança" em seguida um dado eletrônico, que serve para selecionar uma fruta, ao acaso. As opções são armazenadas nas linhas 20 a 70. Qualquer que seja a linha que o computador passe, ele armazena a fruta como um cordão em **B$** e vai para a linha 80. Nesse ponto você tem que adivinhar qual fruta foi sorteada e digitar seu palpite, que será armazenado em **A$** e comparado com **B$**. Se estes forem iguais, o computador imprimirá: "Você acertou na primeira!", ou "Você tentou (tantas) vezes até acertar...". Se **A$** não for igual a **B$** o computador imprimirá: "Você errou"; nesse caso, uma nova tentativa deve ser feita.

Mesmo que você adivinhe o nome da fruta, a condição **A$ = B$** não será satisfeita se você escrevê-lo de modo incorreto. Para que o computador considere dois cordões iguais, eles devem ser graficamente idênticos — isto é, letras, espaços, sinais de pontuação e números devem ser iguais.

Uma das funções dessa técnica é verificar entradas:

```
IF A$= "SIM" THEN PRINT "VOCÊ
TEM CERTEZA?"
```

A condição não será satisfeita se a palavra "sim" for digitada com letras minúsculas.

ORDENAÇÃO DE CORDÕES

As cadeias de caracteres também podem ser comparadas utilizando-se os sinais de desigualdade < e > em linhas como essa:

```
IF A$<B$ THEN PRINT "O PRIMEIRO
É  ";A$
```

Aqui, a condição **A$<B$** pergunta se o cordão em **A$** vem antes do cordão em **B$**, quando eles são colocados em ordem alfabética. Mas tenha cuidado: o computador coloca cordões em ordem alfabética, examinando o código ASCII de cada uma das letras ou caracteres que compõem o cordão; por exemplo, a letra A no código ASCII equivale a 65 e Z, a 90. O problema é que as letras minúsculas também possuem códigos ASCII: o a é 97 e o z, 122. Assim, todos os cordões que começarem com letras maiúsculas serão colocados em primeiro lugar.

Para complicar ainda mais, sinais de pontuação, espaços e outros signos também possuem códigos ASCII; dessa forma é difícil prever em que ordem ficará um cordão constituído por diversos signos.

Com o devido cuidado, os operadores de maior (>) e menor (<), ainda podem ser utilizados para organizar cordões em ordem alfabética. Uma rotina de ordenação alfabética, que aplica a chamada técnica de ordenação por bolhas, será explicada mais adiante numa lição sobre estruturação de programas.

COMO FRACIONAR CORDÕES

É possível também separar um caractere ou uma seqüência de caracteres de um cordão. Nos computadores TRS, MSX e Apple II isso é feito utilizando-se funções **LEFT$**, **RIGHT$** e **MID$**. Já o Spectrum aplica uma técnica diferente.

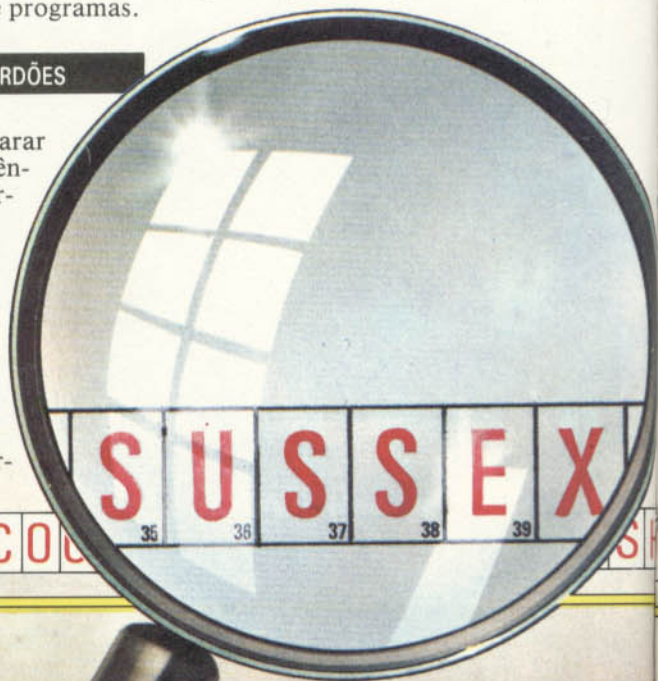
A função **LEFT$ (A$, número)** toma caracteres a partir do primeiro à esquerda no cordão **A$** e fornece tantos caracteres quantos forem especificados no número entre parênteses. Se **A$** for "SR MARIO SILVA" e você especificar dois caracteres — **LEFT$ (A$,2)** —, o resultado será "SR".

De igual modo, o comando **RIGHT$** conta a partir da outra ponta do cordão — ou seja, de seu final à direita. Assim, **RIGHT$ (A$,5)** produzirá "SILVA".

Com o **MID$** você pode especificar dois parâmetros: a posição inicial no cordão, e o número de caracteres a serem extraídos. Por exemplo, **MID$ (A$,4,5)** começará no quarto caractere **M** e pegará cinco caracteres, produzindo "MARIO". Se for especificado somente um número, tal como em **MID$(A$,4)**, a função retornará todos os caracteres daquela posição, até o final do cordão.

O método empregado pelos micros da linha Sinclair, tais como o ZX-81 e o ZX Spectrum, é mais simples e direto. Nesses computadores, existe apenas uma função para manipulação de cordões: o comando **TO**. Por exemplo, **A$ (número TO número)**. **A$** identifica o cordão a ser fracionado e os dois parâmetros correspondentes ao início e ao fim do sub-cordão a ser extraído.

Usando o mesmo cordão do exemplo anterior, **A$** = "SR MARIO SILVA", **A$(1 TO 2)** dá a você "SR", **A$ (4 TO 8)** fornece "MARIO" e **A$(10 TO 14)** produz "SILVA". Não é necessário especificar os números na função acima. Se for omitido o primeiro, o sub-cordão a ser extraído se iniciará no primeiro caractere. Se o segundo for omitido, a função extrairá até o final do cordão.

Eis aqui um programa que utiliza todas as funções explicadas acima. É um jogo de palavras para duas pessoas.

ES DETYPIST AGENCYZ CO

Uma delas entra uma expressão cujas letras serão embaralhadas e impressas pelo computador como um anagrama para o segundo jogador resolver.

```
10 CLS
20 PRINT @75,"ANAGRAMA"
30 PRINT @161,"DIGITE A PALAVRA
 A SER EMBARA   LHADA"
40 A$=INKEY$:IF A$="" THEN 40
43 IF A$=CHR$(13) THEN 55
46 IF A$<" " THEN 40
49 W$=W$+A$:GOTO 40
55 WORD$=W$
70 CLS
80 FOR N=LEN(W$)TO 1 STEP -1
90 M=RND(N)
100 A$=A$+MID$(W$,M,1)
110 W$=LEFT$(W$,M-1)+MID$(W$,M+
1)
120 NEXT N
130 PRINT @65,"O ANAGRAMA E "A$
140 PRINT @129,"QUE PALAVRA VOC
E ACHA QUE E?"
160 INPUT GUESS$
170 G=G+1
180 IF GUESS$<>WORD$ THEN PRINT
" ERRADO, TENTE OUTRA VEZ":GOTO
 160
190 PRINT:PRINT" MUITO BEM !"
200 IF G=1 THEN PRINT" VOCE USO
U 1 TENTATIVA" ELSE PRINT"VOCE
USOU";G;"TENTATIVAS"
210 PRINT @480,"QUER JOGAR DE N
OVO (S/N)?"
220 A$=INKEY$:IF A$<>"S" AND A$
<>"N" THEN 220
230 IF A$="S" THEN RUN
240 END
```

```
 10 CLS : LET a$="": LET g=0
 20 PRINT '"           ANAGRAMA"
 30 PRINT ''"DIGITE A PALAVRA
 A SER EMBARALHADA"
 40 POKE 23609,20: POKE 23658,
 8: POKE 23624,63
 50 INPUT w$
 55 LET s$=w$
 60 POKE 23624,56
 70 CLS
 80 FOR n=LEN w$ TO 1 STEP -1
 90 LET m=INT (RND*n)+1
100 LET a$=a$+w$(m)
110 LET w$=w$( TO m-1)+w$(m+1
TO )
120 NEXT n
130 PRINT '"O ANAGRAMA E ";a$
140 PRINT '"QUE PALAVRA VOCE A
CHA QUE E?"
160 INPUT  LINE g$
170 LET g=g+1
180 IF g$<>s$ THEN  PRINT "ERR
ADO,TENTE OUTRA VEZ": GOTO 160
190 PRINT '"MUITO BEM!"
195 IF g=1 THEN  PRINT "VOCE P
RECISOU DE UMA TENTATIVA!":
GOTO 210
200 PRINT "VOCE PRECISOU DE ";
g;" TENTATIVAS"
210 PRINT '"QUER JOGAR OUTRA V
EZ(S/N)?"
220 LET a$=INKEY$: IF a$<>"S"
AND a$<>"N" THEN  GOTO 220
230 IF a$="S" THEN  RUN
```

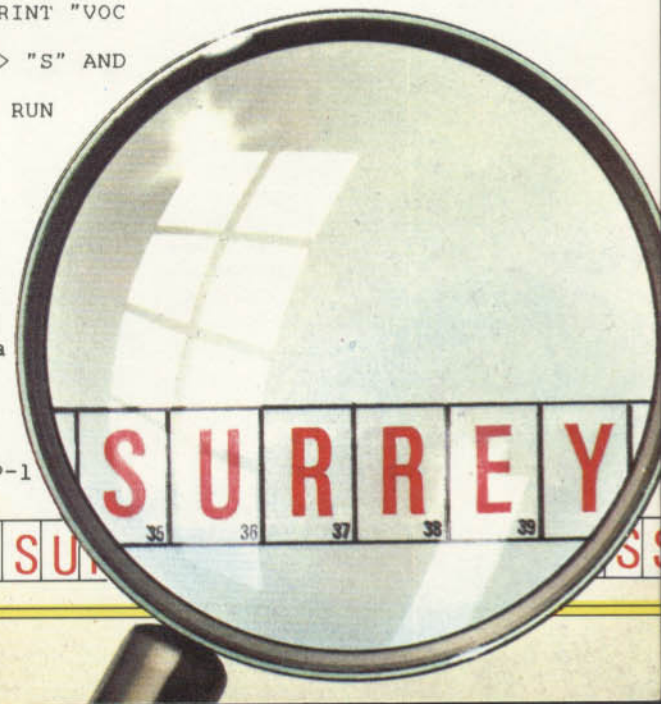
```
10  HOME
20  HTAB 5: VTAB 2: PRINT "PROG
RAMA DE ANAGRAMAS"
30  VTAB 5: PRINT "DIGITE UMA P
ALAVRA PARA SER MISTURADA:"
50  INPUT W$
55 WO$ = W$
70  HOME
80  FOR N =  LEN (W$) TO 1 STEP
 - 1
90 M =  INT ( RND (1) * (N - 1)
) + 1
100 A$ = A$ +  MID$ (W$,M,1)
105  IF M = 1 THEN W$ =  MID$ (
W$,M + 1): GOTO 120
110 W$ =  LEFT$ (W$,M - 1) +  M
ID$ (W$,M + 1)
120  NEXT
130  HTAB 4: VTAB 12: PRINT "O
ANAGRAMA E: ";A$
140  PRINT : HTAB 4: PRINT "QUE
 PALAVRA E ESTA ";
160  INPUT GUESS$
170 G = G + 1
180  IF GUESS$ <  > WO$ THEN  P
RINT : PRINT "ERRADO! TENTE NOV
AMENTE.": GOTO 160
190  PRINT : PRINT "MUITO BEM!"

200  IF G = 1 THEN  PRINT "VOCE
 ACERTOU NA PRIMEIRA!": GOTO 21
0
205  PRINT "VOCE TENTOU ";G;" V
EZES."
210  PRINT : PRINT : PRINT "VOC
E JOGA OUTRA VEZ? ";
220  GET A$: IF A$ <  > "S" AND
 A$ <  > "N" THEN 220
230  IF A$ = "S" THEN  RUN
240  END
```

```
10 CLS:R=RND(-TIME)
20 LOCATE 5,2:PRINT"
PROGRAMA DE ANAGRAMAS"
30 LOCATE 1,5:PRINT"
Digite uma palavra para
ser misturada:"
50 INPUT W$
55 WO$=W$
70 CLS
80 FOR N=LEN(W$)TO1 STEP-1
90 M=INT(RND(1)*(N-1))+1
100 A$=A$+MID$(W$,M,1)
110 W$=LEFT$(W$,M-1)+MID$(W$,M+
1)
120 NEXT
130 LOCATE 4,12:PRINT"O anagram
a é: ";A$
140 LOCATE 4,14:PRINT"Que palav
ra é esta? "
160 INPUT GUESS$
170 G=G+1
180 IF GUESS$<>WO$ THEN PRINT "
Errado! Tente novamente.":GOTO
160
190 PRINT:PRINT"Muito bem!"
200 IF G=1 THENPRINT"Você acert
ou na primeira!" ELSE PRINT"Voc
ê tentou";G;"vezes."
210 PRINT:PRINT:PRINT"Você joga
 outra vez? (S/N)"
220 A$=INKEY$:IF A$<>"S" AND A$
<>"N" THEN 220
230 IF A$= "S" THEN RUN
240 END
```

Quando rodar esse jogo, você verá que a palavra digitada em primeiro lugar não aparecerá na tela. Esse ocultamento tem por objetivo evitar que o seu adversário veja qual é a palavra. Cada computador utiliza um método diferente para fazer isso. Os micros da linha Spectrum imprimem a palavra na mesma cor do plano de fundo, de modo a torná-la invisível.

O TRS-Color entra a palavra utilizando **INKEY$**, que apanha um caractere por vez no teclado, sem imprimi-los na tela. O MSX e o Apple II simplesmente apagam a tela depois da digitação da palavra.

A rotina para embaralhar as letras está nas linhas 80 a 120. O que acontece é que os caracteres são apanhados aleatoriamente na palavra e acrescentados

(um de cada vez) a **A$** que, gradualmente, monta o anagrama.

A linha 90 escolhe um número aleatório M, entre 1 e a extensão da palavra. A linha 100 pega a M-ésima letra e a acrescenta a **A$**. A linha 110 remove esse caractere da expressão original, tomando a parte esquerda da palavra até a posição M (exclusive) e acrescentando o que vier após M. Agora, a expressão terá um caractere a menos, mas da próxima vez, ao fechar o laço, a variável **N** também terá seu valor reduzido de um; assim, o número aleatório M será novamente limitado pela extensão da palavra.

Quando todos os caracteres tiverem sido extraídos, o anagrama será impresso na tela e o seu adversário terá que adivinhar qual a palavra original. Quando ele acertar, o computador informará sobre o número de tentativas feitas durante o jogo e oferecerá uma nova partida.

É bastante fácil alterar esse programa, de modo a fazer com que as palavras sejam lidas a partir de uma lista de dados em linhas **DATA**, em vez de entrá-las separadamente a todo momento. Você pode igualmente elaborar um sistema de contagem de pontos, atribuindo, por exemplo, dez pontos para uma adivinhação correta feita na primeira tentativa, nove para uma adivinhação correta após duas tentativas e assim por diante.

Uma outra utilização desse método de fracionamento é a manipulação de datas. Estas formam um cordão, mesmo quando são digitadas em forma numérica — como, por exemplo, 27/03/51, que significa 27 de março de 1951. É bem verdade que você não pode manipular uma data expressa desse modo pelas leis normais da matemática. Mas pode trabalhar com diferentes partes de uma data utilizando aritmética.

Dessa maneira, é possível, por exemplo, calcular quantos anos uma pessoa tem em um determinado dia, desde que se conheça a data de seu nascimento. Da mesma forma, pode-se calcular quantos dias se passaram entre duas datas determinadas.

Os comandos **LEFT$**, **RIGHT$** e **MID$** — ou os métodos equivalentes usados pelos micros da linha Sinclair — podem ser usados facilmente para separar o dia, o mês e o ano de uma data. Caso seja empregada a função **VAL** (explicada mais adiante), os cordões contendo números poderão ser convertidos para números, que podem ser manipulados por meio de operações aritméticas tais como divisão, soma, subtração, etc.

O TAMANHO DE UM CORDÃO

Às vezes é útil conhecer o comprimento de um cordão. Se, por exemplo, você tiver um espaço limitado de memória, reservado para a informação digitada no teclado, ou se dispuser de uma área restrita na tela para exibi-la, pode ser conveniente verificar a extensão da entrada antes de continuar com o programa.

O comando **LEN(A$)** fornece o número de caracteres de um cordão **A$**. É uma função numérica e pode ser manipulada de acordo com as leis da álgebra. Por exemplo, se **A$ = "SR JOÃO SILVA", LEN (A$) = 13**. Mas, se o programa exigir que o usuário entre um nome, e só houver espaço suficiente na tela para exibir onze letras, o trecho seguinte poderá ser utilizado para informar ao usuário que ele deve limitar a extensão do nome a ser entrado, eliminando letras ou mesmo palavras:

```
10 PRINT "DIGITE O NOME DO
ASSUNTO"
20 INPUT A$
30 IF LEN(A$)>11 THEN PRINT
"APENAS 11 CARACTERES
DISPONIVEIS";:GOTO 10
```

O usuário deverá então reduzir sua entrada e teclar "JOÃO SILVA".

Em outros casos, pode ser mais fácil truncar automaticamente uma entrada. Isso é feito com uma linha tal como a que se segue:

```
IF LEN(A$)>15 THEN LET A$=
LEFT$(A$,15)
```

```
IF LEN(A$)>15 ThEN
LET A$=A$(TO 15)
```

NÚMEROS

Uma das aplicações mais importantes das funções de conversão de cordões para números consiste em fazer programas "à prova de idiotas", ou seja, que impeçam a digitação equivocada dos dados. Por exemplo, na seção abaixo:

```
100 PRINT "DIGITE UM NUMERO"
110 INPUT A
```

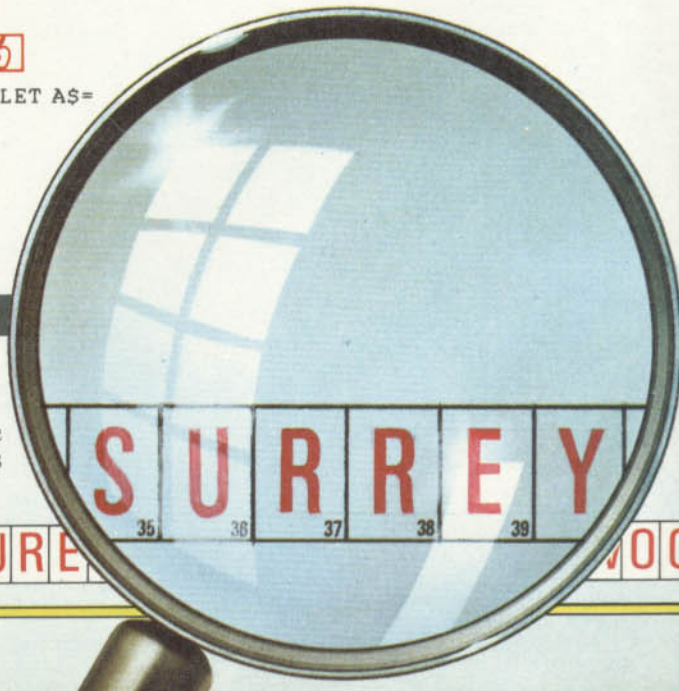
...o computador espera que o usuário digite um número. Se este digitar um caractere não numérico, a maioria dos computadores imprimirá uma mensagem de erro. Esta, porém, avisará apenas que alguma coisa está errada, sem dizer onde está o erro.

Porém, se for utilizado:

```
110 INPUT A$
```

...o computador aceitará qualquer coisa digitada pelo usuário sem emitir mensagens de erro. O passo seguinte será converter o cordão em um número. Para isso, utiliza-se a função **VAL( )**. Na maioria dos computadores, se houver alguma letra misturada com números no cordão de entrada, **VAL** retornará um valor igual a 0.

Infelizmente, esse método não serve para os micros da linha Sinclair, nos quais o emprego do comando **VAL** com um cordão de caracteres que contenha letras produz uma mensagem de erro. Nesses micros, a função **VAL** só deve ser usada se o cordão for constituído apenas de números. Assim, **VAL("1984")** fornecerá **1984**; mas **VAL("26/10/84")** fornecerá 0.03095..., porque os micros da linha Sinclair utilizam o **VAL** para avaliar a expressão **26 : 10 : 84**. Por esse motivo, os possuidores do Sinclair não precisam ler o resto desta seção, dirigindo-se diretamente para o próximo "capítulo" do artigo (*De números a cordões*).

Nos computadores TRS-80, TRS-

Color, Apple e MSX, a função **VAL** extrai a parte numérica do cordão. Assim, se **A$** contiver apenas dígitos numéricos, **VAL(=A$)** retornará o valor numérico correspondente, que poderá ser posteriormente utilizado no programa. Se, ao contrário, **A$** não contiver números, **VAL(A$)** retornará o valor 0.

Você poderá então escrever uma pequena sub-rotina que explique em detalhes ao usuário que ele cometeu um erro, oferecendo-lhe ao mesmo tempo outra oportunidade de entrar os dados sem interromper o programa ou desarrumar a tela do computador.

Um ponto importante a ser observado em relação ao comando **VAL** é que ele só extrai números no início do cordão. Dessa maneira, **VAL("25 Julho")** é igual a 25, mas **VAL("JULHO 25")** é equivalente a 0.

A função **VAL** é útil também para ordenar números extraídos de um cordão. Se, por qualquer razão, você tiver os nomes e as notas de uma classe de alunos em um cordão como **A$="32 CARLOS"**, **B$"45 MARCOS"** e **C$="41 JOSÉ"** e assim por diante, e quiser uma média dos resultados, o primeiro passo a ser dado será extrair as notas individuais utilizando o **VAL**. Assim, **VAL(=A$)** fornece **32**, **VAL(=B$)** extrai **45** e **VAL (C$)** separa **41**.

Do mesmo modo, o comando **VAL** pode ser utilizado para ignorar as unidades de medida digitadas junto com os valores. O próximo programa mostrará a você a diferença:

```
100 LET A$="32KG"
110 LET B$="110KG"
120 PRINT"A$+B$= ";A$+B$
130 PRINT"VAL(A$)+VAL(B$)=";VAL
(A$)+VAL(B$)
140 END
```

DE NÚMEROS A CORDÕES

A função **STR$** desempenha um papel oposto ao do comando **VAL**: enquanto este extrai números de cordões, aquela converte números em cordões. A vantagem dessa operação é que os cordões podem ser, por exemplo, fracionados e concatenados.

O comando **STR$** possui, portanto, inúmeras aplicações.

O programa a seguir transforma um número decimal em binário. Embora os computadores trabalhem diretamente com a aritmética de números binários, a linguagem BASIC é capaz de manipular apenas decimais ou, em alguns casos, hexadecimais e octais. Assim, quando um número binário aparecer em um programa em BASIC, ele precisa ser tratado como se fosse um cordão:

```
10 PRINT"DECIMAL PARA BINARIO"
20 PRINT"DIGITE  NUMERO DECIMA
L INTEIRO"
30 INPUT D
40 LET B$=""
50 LET B$=STR$(D-INT(D/2)*2)+B$
60 LET D=INT(D/2)
70 IF D<>0 THEN GOTO 50
80 PRINT"O NUMERO BINARIO E";B$
```

Quando for entrado um número decimal positivo, a linha 40 igualará **B$** a um cordão vazio ou nulo; este será então progressivamente preenchido com dígitos obtidos pelo processamento nas linhas 50 a 80.

Na verdade, a linha 50 é a que monta o número binário. Ela subtrai duas vezes o valor inteiro da metade do valor decimal, a partir do próprio número decimal. Essa é uma maneira de testar se o número é ímpar ou par. Se for ímpar, o resultado da operação será 1; se for par, o resultado será 0. Estes são, como você já sabe, os dois dígitos binários.

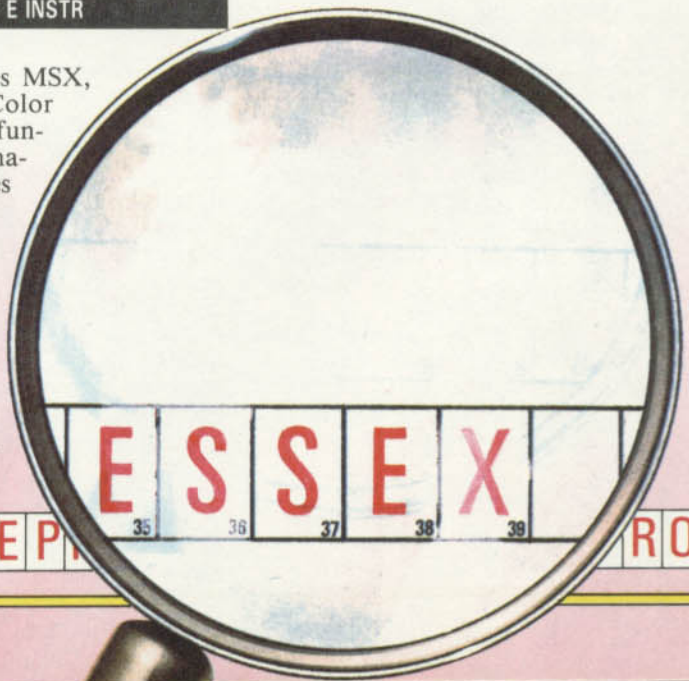
Esses dígitos são convertidos em um cordão mediante o emprego da função **STR$**; em seguida, eles são montados por intermédio da concatenação de cada novo dígito com o resto do cordão.

STRING$ E INSTR

Os computadores MSX, TRS-80 e TRS-Color contam com duas funções adicionais de manipulação de cordões alfanuméricos: **STRING$** e **INSTR** (estas não existem nos micros da linha Apple, TK-2000 e Sinclair). O **STRING$(N,A$)** produz um cordão de **N** caracteres definidos por **A$**. **N** deve ser um número ou uma variável numérica, enquanto **A$** deve ser uma variável alfanumérica, um caractere ou um cordão de caracteres entre aspas. Por exemplo, **PRINT STRING$** (6, "*") imprimirá ******.

Nos microcomputadores das linhas TRS-80 e TRS-Color a função **STRING$** utiliza somente *o primeiro* caractere do cordão especificado, ignorando o resto. Por exemplo, **STRING$(3,"XBC")** imprime XXX, apenas.

Eis aqui dois programas (um para os modelos compatíveis com a linha TRS-Color e o outro para os da linha MSX) que utilizam **STRING$** para imprimir uma borda decorativa na tela. Você pode utilizá-los para abrilhantar a página-título de um jogo:

```
10 CLS0
20 A$=CHR$(158)+STRING$(30,CHR$
(156))+CHR$(157)
30 B$=CHR$(154)+CHR$(174)+STRIN
G$(28,CHR$(172))+CHR$(173)+CHR$
(149)
40 C$=CHR$(154)+CHR$(171)+STRIN
G$(28,CHR$(163))+CHR$(167)+CHR$
(149)
50 D$=CHR$(155)+STRING$(30,CHR$
(147))+CHR$(151)
60 F$=CHR$(154)+CHR$(170)+STRIN
G$(28," ")+CHR$(165)+CHR$(149)
70 PRINT A$;
80 PRINT B$;
90 FOR K=1 TO 11
100 PRINT F$;
110 NEXT K
120 PRINT C$;
130 PRINT D$;
140 PRINT @237,"OLA !";
150 GOTO 150
```

```
10 CLS:COLOR 6,11
20 A$=CHR$(219)+STRING$(36,210)
+CHR$(219)
30 B$=CHR$(209)+CHR$(32)+STRING
$(34,192)+CHR$(32)+CHR$(209)
40 C$=CHR$(209)+CHR$(32)+STRING
$(34,195)+CHR$(32)+CHR$(209)
50 D$=CHR$(209)+CHR$(222)+STRIN
G$(34,32)+CHR$(221)+CHR$(209)
70 PRINTA$
80 PRINTB$
90 FORK=1TO18
100 PRINTD$
110 NEXT
120 PRINTC$
130 PRINTA$
140 LOCATE 16,10:PRINT"OLÁ";
150 GOTO 150
```

Os programas para o TRS-Color empregam diversos caracteres para imprimir as bordas. Os blocos são impressos com esquema de duas cores — amarelo/preto e azul/preto. Para se fazer uma borda simétrica, as cores são invertidas, de cima para baixo, e da esquerda para a direita. Cinco cordões são utilizados: **A$** e **B$** definem a parte superior da borda, **F$** os lados, e **C$** e **D$** a margem inferior. A linha 150 impede que o sinal de prontidão OK perturbe o resultado final na tela. O programa para o MSX também utiliza caracteres gráficos para fazer as bordas.

COMO PESQUISAR UM CORDÃO

O comando **INSTR** examina o cordão, procurando pela primeira ocorrência de um sub-cordão mais curto. Assim, uma das aplicações mais comuns para o **INSTR** consiste em encontrar uma palavra dentro de uma sentença, ou uma letra dentro de uma palavra.

Para usá-lo, especifica-se **INSTR(A$, B$)**: isto significa que a função deve indicar a posição inicial do sub-cordão **B$** no cordão **A$**. Por exemplo, **PRINT INSTR("ALO", "L")** exibirá o número 2. Se o computador não conseguir achar o cordão, ele retornará 0:

```
10 A$="ALO"
20 B$="W"
30 PRINT INSTR(A$,B$)
```

Em alguns micros é possível ainda especificar um número que define a posição no cordão maior, a partir da qual a função **INSTR** funcionará. Assim, em **INSTR(P,A$,B$)**, essa posição é indicada por **P**. Essa forma de utilização é muito útil quando se deseja procurar ocorrências repetidas do mesmo sub-cordão no cordão principal.

Suponhamos que se deseja encontrar todas as ocorrências da letra I na palavra INCONSTITUCIONAL. O primeiro I está na posição 1. Para achar o segundo, entretanto, devemos usar o parâmetro P, de modo a começar a busca a partir de **P=1+1** (ou seja, P igual a 2); portanto, **INSTR(P,"INCONSTITUCIONAL","I")**. Para localizar a terceira ocorrência do I, o número **P** deve ser incrementado mais uma vez, e assim por diante, até que um resultado 0 seja retornado pela função.

Esse comando serve ainda para verificar uma entrada feita pelo teclado. Se for preciso escolher uma opção em um menu, digitando a sua letra inicial, as alternativas poderão ser:

```
10 PRINT "(I)MPRIMIR O TEXTO"
20 PRINT "(G)RAVAR O TEXTO"
30 PRINT "(C)ARREGAR NOVO
TEXTO"
40 PRINT "(E)DITAR O TEXTO"
50 PRINT "ESCOLHA UMA OPÇAO"
60 INPUT A$
```

O modo mais fácil de se verificar a validade de uma letra digitada pelo usuário consiste em acrescentar a linha a seguir:

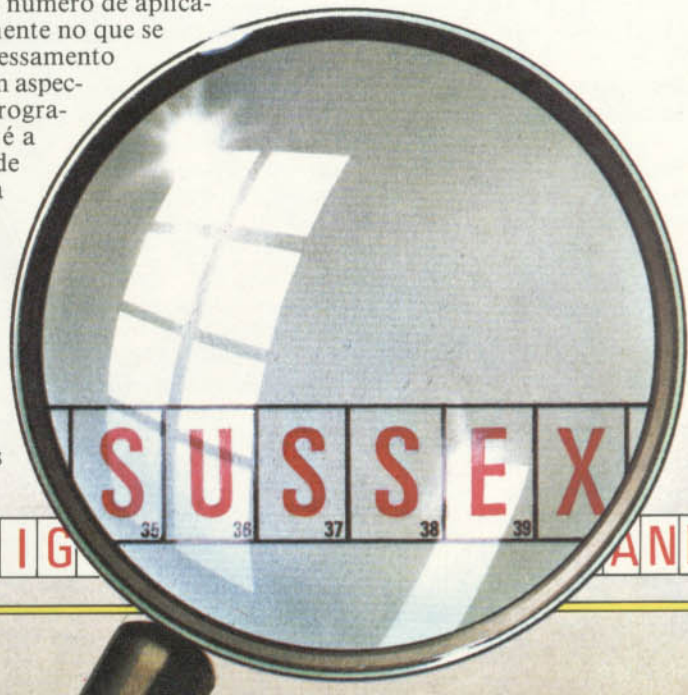
```
70 IF INSTR("PSLE",A$)=0 THEN
GOTO 50
```

Isso elimina possíveis mensagens de erro, que "estragariam" a tela.

PROCESSAMENTO DE PALAVRAS

As funções alfanuméricas do BASIC têm um grande número de aplicações, especialmente no que se refere ao processamento de palavras. Um aspecto comum a programas desse tipo é a possibilidade de substituir uma determinada palavra por outra (isso pode ser necessário, por exemplo, quando aparecer um erro de ortografia em todas as ocorrências de uma palavra). O **INSTR** é usado para localizar essas ocorrências. Se a nova palavra tiver um comprimento diferente daquela a ser substituída, será necessário movimentar o resto do texto para criar espaço.

```
10 LINEINPUT "DIGITE TEXTO: ";T
$
20 LINEINPUT "PALAVRA A SER TRO
CADA? ";W$
30 LINEINPUT "NOVA PALAVRA? ";N
W$
40 P=1
50 PO=INSTR(P,T$,W$)
60 IF PO=0 THEN GOTO 100
70 T$=LEFT$(T$,PO-1)+NW$+RIGHT$
(T$,LEN(T$)-PO-LEN(W$)+1)
80 P=PO+LEN(NW$)
90 GOTO 50
100 PRINT T$
110 GOTO 20
```

```
10  INPUT "DIGITE UMA FRASE ";T
$
15 T$ =  CHR$ (32) + T$ +  CHR$
 (32)
20  INPUT "PALAVRA A SER SUBSTI
TUIDA ";W$
30  INPUT "NOVA PALAVRA ";NW$
35 P = 0
40 P = P + 1
50 A$ =  MID$ (T$,P, LEN (W$))
60  IF A$ <  > W$ THEN 90
70 T$ =  LEFT$ (T$,P - 1) + NW$
 +  RIGHT$ (T$, LEN (T$) - P -
 LEN (W$) + 1)
80 P = P +  LEN (NW$) - 1
90  IF P <  LEN (T$) THEN 40
100  PRINT T$
110  GOTO 20
```

```
 10 INPUT "DIGITE O TEXTO ";
 LINE t$: PRINT t$
 20 INPUT "PALAVRA A SER CORRI
 GIDA "; LINE w$: LET w=LEN w$
 30 INPUT "NOVA PALAVRA ";
 LINE n$: LET n=LEN n$
 35 LET p=0
 40 LET p=p+1
 50 IF p+w-1>LEN t$ THEN  GOTO
  100
 60 IF t$(p TO p+w-1)<>w$ THEN
   GOTO 40
 70 LET t$=t$( TO p-1)+n$+t$(p
 +w TO ): GOTO 40
100 PRINT t$
110 GOTO 20
```

```
10 LINEINPUT"Digite uma frase:
";T$
20 LINEINPUT"Palavra a ser subs
tituída? ";W$
30 LINEINPUT"Nova palavra? ";NW
$
40 P=1
50 PO=INSTR(P,T$,W$)
60 IFPO=0THEN100
70 T$=LEFT$(T$,PO-1)+NW$+RIGHT$
(T$,LEN(T$)-PO-LEN(W$)+1)
80 P=PO+LEN(NW$)
90 GOTO50
100 PRINTT$
110 GOTO20
```

Comece entrando sentenças bem simples e tente o procedimento de substituição de algumas letras ou palavras. Quando estas forem curtas como "ou", "na", etc., será necessário digitá-las com um espaço antes e outro depois; caso contrário, toda ocorrência de "ou", "na", etc., dentro de outras palavras, (por exemplo, "ouro" e "banana") também será modificada.

O programa funciona assim: a linha 40 define o **P** para começar o processo de pesquisa no início do texto. As linhas 50 e 60 encontram a primeira ocorrência da palavra ou da letra que se deseja substituir; em seguida, a linha 70 a substitui por uma nova palavra. Essa linha investiga o texto do início até a palavra a ser modificada, acrescenta a nova expressão e concatena o restante do texto. O processo se repete até que todas as ocorrências da palavra tenham sido substituídas. A linha 100 imprime o resultado (os programas profissionais de processamento de palavras são, evidentemente, mais complexos do que nosso exemplo).

ASSEMBLER PARA O SPECTRUM

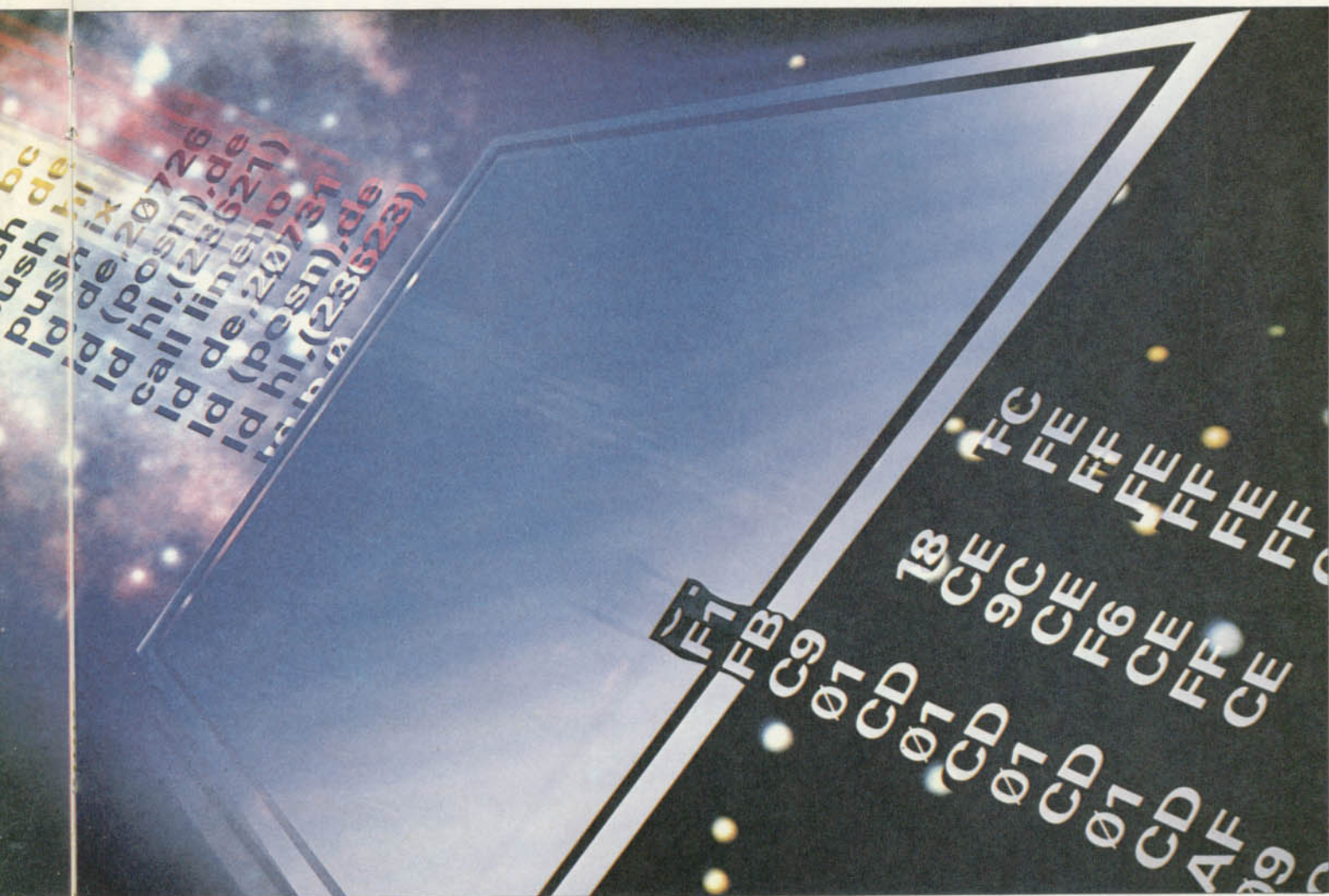
Um programa que calcula a dimensão dos saltos e faz a tradução dos códigos mnemônicos Assembly para o sistema hexadecimal: o que mais poderíamos desejar na vida?

Montar um programa em linguagem de máquina calculando os códigos a mão pode ser bastante cansativo, mesmo que se saiba de cor os códigos mnemônicos e seus equivalentes hexadecimais e se esteja familiarizado com todos os modos de endereçamento. Com efeito, além de estar sujeito a erro, o trabalho de tradução e de transferência do programa para o computador é sempre tedioso. A solução está em entregar esse trabalho ao próprio computador.

Nesta lição apresentamos um programa Assembler para os micros compatíveis com o Sinclair Spectrum (por exemplo, o TK-90X) com um mínimo de 48K de memória. Versões para outros computadores serão abordadas nas próximas lições desta série. O TK-2000 não necessita de um programa desses, pois já vem com um míni-Assembler. O programa apresentado aqui não funciona no Spectrum de 16K, nem nos modelos compatíveis com o ZX-81. Ele está escrito em BASIC, o que o torna bem mais lento que os Assemblers convencionais, elaborados em código de máquina. O importante, porém, é que ele não só funciona, como é capaz de montar programas Assembly, publicados por nós ou obtidos de outras fontes. Um programa longo levará um bom tempo para ser montado.

O ASSEMBLER

```
5000 DIM k$(110,4): DIM k(110):
 DIM m(110): LET h$="0123456789
ABCDEF": LET b$="": LET g$="012
3456789abcdef"
5010 DIM t$(110,24): DIM r(100)
: DIM z$(100,6): DIM z(100)
5020 DIM b(9): LET b(1)=1: FOR
i=2 TO 9: LET b(i)=b(i-1)+b(i-1
): NEXT i
5030 DIM r$(8,4,4): FOR j=1 TO
4: FOR i=1 TO 8: READ r$(i,j):
NEXT i: NEXT j
5040 DATA "0","1","2","3","4","
5","6","7","nz","z","nc","c","p
o","pe","p","m","0","8","16","2
4","32","40","48","56","hl","ix
","iy","bc","de","hl","sp"," "
5050 DIM s$(8,2,4): DIM t(18):
DIM u$(18,10): FOR j=1 TO 2: FO
R i=1 TO 8/j: READ s$(i,j): NEX
T i: NEXT j: FOR j=1 TO 18: REA
D t(j),u$(j): NEXT j
5060 DATA "b","c","d","e","h","
l","(hl)","a","bc","de","hl","s
p",235,"de",8,"af",227,"(sp)",6
0742,"0",60758,"1",60766,"2",23
3,"(hl)",56809,"(ix)",65001,"(i
y)",10,"(bc)",26,"(de)",60767,"
r",2,"(bc)",18,"(de)",60751,"r"
,249,"sp",60743,"i",60759,"i"
5070 DEF FN b(x,i)=INT (x/b(i+1
))-INT (x/b(i+2))*2
5080 DEF FN x(x,i)=x-b(i+2)*FN
b(x,i)+b(i+1)
5090 DEF FN j(x,i)=INT x-b(i+1)
*INT (x/b(i+1))
5100 DEF FN e(i$,j$)=(i$=j$( TO
 LEN (i$)))
5110 FOR i=1 TO 110: READ k$(i)
,k(i),m(i): IF NOT FN e("*",k$(
i)) THEN  NEXT i
5120 DATA "ld",10,10,"ld",26,10
,"ld",60767,10,"ld",60759,10,"l
d",2,138,"ld",18,138,"ld",60751
,138,"ld",60743,138,"ld",64,6,"
ld",50,202,"ld",58,74,"ld",249,
139,"ld",34,195,"ld",42,67,"ld"
,60779,197,"ld",60771,199,"ld",
97,165,"ld",64,54
5130 DATA "adc",136,50,"adc",60
746,3,"add",128,50,"add",9,149,
"and",160,48,"or",176,48,"xor",
168,48,"nop",0,0,"sub",144,48,"
sbc",152,50,"sbc",60738,3,"cp",
184,48,"jp",130,45,"jp",233,9,"
jp",56809,9,"jp",65001,9,"jp",1
31,49,"jr",96,45,"jr",88,41
5140 DATA "call",132,45,"call",
141,41,"ret",201,0,"djnz",74,40
,"dec",11,17,"dec",5,16,"inc",3
,17,"inc",4,16,"push",197,17,"p
op",193,17,"di",243,0,"ei",251,
0,"halt",118,0,"ex",235,139,"ex
",8,15,"ex",227,143,"exx",217,0
5150 DATA "rst",199,132,"rts",1
92,5,"bit",52032,20,"defb",-256
,40,"ccf",63,0,"scf",55,0,"cpl"
,47,0,"cpd",60841,0,"cpdr",6085
7,0,"cpi",60833,0,"cpir",60849,
0,"daa",39,0,"im",60742,8,"im",
60758,8,"im",60766,8
5160 DATA "in",60736,130,"in",1
49,42,"ind",60848,0,"indr",6081
0,0,"ini",60840,0,"ldd",60840,0
,"lddr",60856,0,"ldi",60832,0,"
ldir",60848,0,"neg",60740,0,"ot
dr",60859,0,"otir",60851,0,"out
",60737,2,"out",141,170,"outd",
60843,0,"outi",60835,0
5170 DATA "res",52096,20,"reti"
,60749,0,"retn",60741,0,"rl",51
984,64,"rla",23,0,"rlc",51968,1
6,"rlca",7,0,"rld",60783,0,"rr"
,51992,64,"rra",31,0,"rrc",5197
6,16,"rrca",15,0,"rrd",60775,0
5180 DATA "set",52160,20,"sla",
52000,16,"sra",52008,16,"srl",5
2024,16,"defw",-256,41
5190 DATA "*",0,0: LET ii=i: LE
T k(110)=ii
5200 LET b=0: LET ba=PEEK 23635
+256*PEEK 23636+4: LET n=1
5210 LET cc=1: IF PEEK ba<>234
THEN  LET n=n-1: GOTO 5250
5220 LET cc=cc+1: LET ba=ba+1:
IF PEEK ba=13 THEN  LET ba=ba+5
: LET n=n+1: GOTO 5210
5230 LET t$(n,cc)=CHR$ PEEK ba
```

- TRADUÇÃO DE ASSEMBLY PARA CÓDIGO DE MÁQUINA
- COMO CALCULAR SALTOS E DESVIOS
- O USO DE RÓTULOS
- RESERVE ESPAÇO PARA DADOS E VARIÁVEIS
- COMO COLOCAR OS CÓDIGOS NA MEMÓRIA USANDO **POKE**
- TESTE O PROGRAMA

```
5240 GOTO 5220
5250 FOR g=1 TO 100: LET r(g)=g
-1: NEXT g: LET fh=100
5300 LET k0=0: LET k9=99: LET p
0=0: LET vv=0
5310 LET k=k0: LET p=p0
5320 GOSUB 8000
5330 GOSUB 7000: LET o$=i$: IF
o$(1)="*" THEN  PRINT o$;: GOTO
 5320
5340 IF o$="end" THEN  PRINT '"
   endereco final ";p-1
5350 IF o$="end" THEN  LET p0=p
: GOTO 9999
5370 IF o$<>"org" THEN  GOTO 54
00
5380 GOSUB 7000: LET s=0: IF i$
(1)="*" THEN  LET s=p: LET i$=i
$(2 TO )
5390 LET p=VAL i$+s: PRINT '"
  org ";p;: GOTO 5320
5400 IF p=0 THEN  PRINT "(falta
 org)": LET p=50000
5410 LET p$=o$+"!": FOR i=1+18*
(o$<>"ld") TO 110: IF o$<=k$(i)
 AND p$>k$(i) THEN  GOTO 5500
5420 NEXT i: PRINT o$
5430 IF i$(1)="." THEN  LET i$=
i$(2 TO )
5440 GOSUB 9000: LET gg=r(g)
5450 IF gg<=100 THEN  LET s=SGN
 z(gg): LET b=INT (ABS z(gg)/65
536): LET r=ABS z(gg)-b*65536:
LET q=PEEK r+256*PEEK (r+1): PO
KE r,FN j(p*s+q,8): PRINT " POK
E ";r;" com ";FN j(p*s+q,8): IF
 b THEN  POKE (r+1),FN j((p*s+q
)/256,8): PRINT " POKE ";r+1;"
com ";FN j((p*s+q)/256,8)
5460 IF gg<=100 THEN  LET gh=r(
gg): LET r(gg)=fh: LET fh=gg: L
ET gg=gh: GOTO 5450
5470 IF i$="" THEN  LET r(g)=p+
100: GOTO 5330
5480 PRINT " (Linha nao foi rec
onhecida)"
5490 GOTO 5420
5500 LET z=0: LET r=0: LET e=0:
 PRINT "     ";o$;
5510 LET op=k(i): IF m(i)=0 THE
N  GOTO 6090
```

```
5520 GOSUB 7000: LET a$=i$: PRI
NT " ";a$;
5530 LET m=m(i): LET op=k(i): L
ET b=FN b(m,0): LET b7=b+2*FN b
(m,7)+1: LET z=0: IF FN j(m,3)<
2 THEN  LET c$=a$: GOTO 5720
5540 FOR j=1 TO LEN a$: IF a$(j
)="," THEN  GOTO 5580
5550 NEXT j: IF o$="rst" OR o$=
"rts" THEN  GOTO 5580
5560 IF FN e(o$,k$(i+1)) THEN
LET i=i+1: GOTO 5530
5570 PRINT " (deve haver dois o
perandos)": GOTO 5320
5580 LET b$=a$( TO j-1): LET c$
=a$(j+1 TO )
5590 IF FN b(m,2) THEN  GOTO 56
50
5600 IF FN b(m,7) THEN  LET d$=
c$: LET c$=b$: LET b$=d$
5610 IF b$="ahl"(b+1 TO b+b+1)
THEN  GOTO 5720
5620 IF b$="(c)" AND (o$="in" O
R o$="out") THEN  GOTO 5720
5630 IF (FN e(o$,k$(i+1))) AND
(FN j(m(i+1),3)>=2) THEN  LET i
=i+1: GOTO 5530
5640 PRINT "(primeiro operando
deve ser a ou hl)": GOTO 5320
5650 IF FN b(m,1) THEN  GOTO 56
90
5660 LET e$=(b$+"    ")( TO 4):
FOR j=1 TO 8: IF e$=r$(j,b7) TH
EN  LET op=op+8*(j-1)*(b7<4)+16
*(j-6)*(b7=4)*(j>3): LET z=(j-1
)*(b7=4)*(j<=3): GOTO 5710
5670 NEXT j: IF p$>k$(i+1) AND
(FN j(m(i+1),3)>=2) THEN  LET i
=i+1: GOTO 5530
5680 PRINT "(primeiro operando
deve ser bit ou sinalizador)":
GOTO 5320
5690 IF FN b(m,7) THEN  LET d$=
c$: LET c$=b$: LET b$=d$: GOTO
5660
5700 LET x=8: GOSUB 5750: IF e
THEN  GOTO 5730
5710 IF c$="" THEN  GOTO 6090
5720 LET x=1+15*b+7*(op<=6 AND
op>=4 OR b$="(c)"): LET b$=c$:
GOSUB 5750: IF NOT e THEN  GOTO
 6090
5730 IF e=2 OR p$>k$(i+1) AND F
N j(m(i+1),3)=FN j(FN x(m,0),3)
 THEN  LET e=0: LET i=i+1: GOTO
 5530
5740 GOTO 5320
5750 LET r=0: IF FN b(m,4) AND
FN e("("( TO NOT b),b$) THEN  L
ET z2=FN e("ix",b$(2-b TO )+" "
)+2*FN e("iy",b$(2-b TO )+" "):
 IF z2 THEN  LET z=z2: LET e$=b
$( TO LEN b$-NOT b): LET b$="(h
l)"(1+b TO 4-b): LET f$="0"+e$(
4-b TO )
5760 IF FN b(m,3) THEN  GOTO 57
90
5770 LET e$=(b$+"    ")( TO 4):
FOR j=1 TO 8/(b+1): IF e$=s$(j,
b+1) THEN  LET op=op+(j-1)*x: R
ETURN
5780 GOTO 5810
5790 LET j2=9+9*(o$="ld"): FOR
j=j2-8 TO j2: IF k(i)<>t(j) THE
N  GOTO 5810
5800 IF FN e(b$,u$(j)) THEN  RE
TURN
5810 NEXT j: IF b$="af" THEN  I
F FN e("p",o$) THEN  LET op=op+
48: RETURN
5820 IF FN b(m,6) AND FN e("(",
b$) THEN  LET b$=b$(2 TO LEN b$
-1): GOTO 5860
5830 IF FN b(m,5) THEN  LET op=
FN x(op+6*NOT b,6): GOTO 5860
5840 IF p$>k$(i+1) THEN  LET e=
2: RETURN
5850 PRINT "(operando incompati
vel)": LET e=1: RETURN
5860 LET r=65536
5870 LET s=1
5880 IF b$="" THEN  GOTO 6080
5890 LET x$=b$(1): LET d$=b$(2
TO ): IF x$="*" THEN  LET r=r+p
*s: LET b$=d$: GOTO 5870
5900 IF x$="+" THEN  LET b$=d$:
 GOTO 5880
5910 IF x$="-' THEN  LET b$=d$:
 LET s=-s: GO' ) 5880
5920 IF x$="'" THEN  LET r=r+CO
DE d$*s: LET b$=d$(2 TO ): GOTO
 5870
5930 LET q=0: IF x$<>"%" OR d$<
"0" OR d$>"2" THEN  GOTO 5960
5940 IF d$>="0" AND d$<"2" THEN
  LET q=q*2+CODE d$-48: LET d$=
d$(2 TO ): GOTO 5940
5950 LET r=r+q*s: LET b$=d$: GO
TO 5870
5960 IF x$<>"$" OR d$<"0" OR d$
>="g" THEN  GOTO 6000
5970 LET x$=CHR$ CODE d$: FOR g
=0 TO 15: IF x$<>h$(g+1) AND x$
<>g$(g+1) THEN  GOTO 5990
5980 LET q=q*16+g: LET d$=d$(2
TO ): GOTO 5970
5990 NEXT g: LET r=r+q*s: LET b
$=d$: GOTO 5870
6000 IF x$<"a" OR x$>"z" THEN
GOTO 6040
6010 LET i$=b$: GOSUB 9000: IF
i$<>"" THEN  GOSUB 9400
6020 IF r(g)<>23000 AND r(g)>10
0 THEN  LET r=r+(r(g)-100)*s: L
ET b$=i$: GOTO 5870
6030 IF r(g)=23000 OR r(g)<=100
 THEN  LET gh=r(fh): LET r(fh)=
r(g): LET r(g)=fh: LET fh=gh: L
ET z(r(g))=(p+SGN op+(ABS op>25
5)+2*(z>0)+65536*((b OR FN b(m,
6)) AND o$<>"jr"))*s: LET b$=i$
: GOTO 5870
6040 IF x$<"0" OR x$>"9" THEN
LET r=0: GOTO 6070
6050 IF b$>="0" AND b$<":" THEN
  LET q=q*10+CODE b$-48: LET b$
=b$(2 TO ): GOTO 6050
6060 LET r=r+s*q: GOTO 5870
6070 PRINT "(enderecamento inco
```

```
rreto)"
6080 LET r=r-(p+2)*(o$="djnz" O
R o$="jr"): RETURN
6090 PRINT TAB 16;: LET by=p/25
6: GOSUB 6190: LET by=p: GOSUB
6190: GOSUB 6160
6100 IF z THEN  LET by=189+z*32
: GOSUB 6180: GOSUB 6160
6110 IF op>=0 THEN  LET by=op/2
56: GOSUB 6170: GOSUB 6150: LET
 by=op: GOSUB 6180
6120 IF r=0 THEN  GOTO 5320
6130 GOSUB 6160: LET by=r: GOSU
B 6180: IF (b OR FN b(m,6)) AND
 o$<>"jr" THEN  LET by=r/256: G
OSUB 6180
6140 GOTO 5320
6150 IF z AND INT by AND NOT b
THEN  GOSUB 6260: LET by=VAL f$
: GOSUB 6180: LET z=0
6160 PRINT " ";: RETURN
6170 IF INT by<=0 THEN  RETURN
6180 LET by=FN j(by,8): POKE p,
by: LET p=p+1
6190 LET by=FN j(by,8): PRINT h
$(1+INT (by/16));h$(FN j(by,4)+
1);
6200 RETURN
7000 IF k>n THEN  LET i$="end":
 RETURN
7010 LET kl=k9+1: IF k9>=LEN t$
(k) THEN  LET i$="/faltando/":
RETURN
7020 LET k9=kl: IF t$(k,kl)=" "
 THEN  GOTO 7010
7030 IF k9>LEN t$(k) THEN  LET
i$=t$(k)(kl TO ): RETURN
7040 IF t$(k,k9)<>" " THEN  LET
 k9=k9+1: GOTO 7030
7050 LET i$=t$(k)(kl TO k9-1):
RETURN
8000 IF k>0 THEN  IF t$(k)(k9 T
O )>t$(99) THEN  PRINT t$(k)(k9
 TO );
8010 POKE 23692,0: LET k=k+1: L
ET k9=0
8020 PRINT : RETURN
9000 LET x$=""
9010 IF i$<"a" OR i$>"z" THEN
GOTO 9030
9020 LET x$=x$+i$(1): LET i$=i$
(2 TO ): GOTO 9010
9030 IF i$<>"" THEN  RETURN
9400 FOR g=1 TO vv: IF FN e(x$,
z$(g)) THEN  RETURN
9410 NEXT g: LET vv=vv+1: LET z
$(vv)=x$: LET g=vv: LET r(g)=23
000
9420 RETURN
```

COMO FUNCIONA O PROGRAMA

Começando na linha 5000, o programa deixa espaço para seu programa em Assembly, que deve ser colocado em linhas **REM**. Cada instrução deve ser escrita com letras minúsculas e posicionada em uma linha **REM** separada.

Antes de montar o programa, devemos proteger uma área de memória para que o Assembler possa colocar os códigos, usando **CLEAR**. A primeira linha de seu programa deve ser mais ou menos assim:

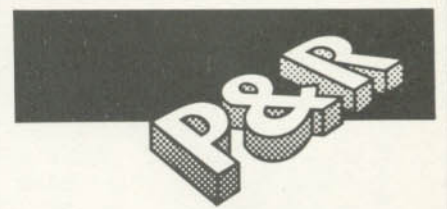
```
10 REM org 55000
```

32000 é o endereço inicial do programa em código. Obviamente, ele deve estar na área protegida da memória.

Se esquecermos de definir **org**, ou seja, a origem, o Assembler colocará o programa a partir de 50000.

O programa aceita os códigos mnemônicos-padrão do chip Zilog Z-80, com exceção dos comandos de retorno condicional. Entretanto, o comando de retorno incondicional de sub-rotinas, cujo mnemônico é **ret**, *funciona* em nosso Assembler.

Algumas vezes, contudo, utilizamos retornos condicionais, como **ret nz**, que significa: "retorne se não for zero". Neste programa, porém, digite a instrução **rts nz**. A razão disso é que **rts** deve substituir **ret** sempre que forem necessários retornos condicionais. Quando se tratar de retorno incondicional, ou seja, quando **ret** não for seguido de nenhuma letra, use o mnemônico padrão **ret**.

Números hexadecimais devem ser

O que acontecerá se houver um erro em meu programa fonte?

Nosso Assembler é capaz de emitir mensagens de erro, pois a sua estrutura de programação está preparada para detectar falhas no programa em Assembly. Alguns comandos, por exemplo, só trabalham com os registros **a** e **hl**. Se tentarmos usá-los com outros registros, o Assembler dirá: "Primeiro operando deve ser **a** ou **hl**"

Se um comando não for reconhecido devido a um erro de digitação, seremos automaticamente informados: "Linha não foi reconhecida". A expressão "Deve haver dois operandos" significa que deixamos de digitar um número vital após o comando. "Operando incompatível" indica um uso inadequado do comando. "Primeiro operando deve ser bit ou sinalizador" significa que um operando inadequado foi usado em um comando de desvio ou de atribuição de bits.

Existe uma maneira de se programar em Assembler nos ZX-81?

Existe; mas o meio para isso não é um programa montador desenvolvido em BASIC, como os que serão apresentados mais adiante para as linhas MSX, Apple II, TRS-80 e TRS-Color.

Montadores em BASIC para o ZX-81 (com seus similares nacionais TK-85, CP-200, etc.) são inviáveis porque o programa resultante seria muito grande e de difícil digitação e operação: na verdade, ele não caberia na memória da maioria dos micros dessa linha; além disso, o processo de tradução e montagem desse programa seria extremamente lento.

Para comprovar isso, observe a listagem para o ZX Spectrum que acompanha este artigo: é a inexistência de declarações **READ** e **DATA** no ZX-81, o que dificulta a programação.

Seria possível programar um Assembler reduzido, sem rótulos alfabéticos, e com um conjunto menor de comandos. Mas isso o tornaria inútil para quem quer aprender a programar seriamente, e impossibilitaria o usuário de digitar e testar os programas em código Assembler que aparecerão em lições futuras. Por isso, quem quiser programar em Assembler no ZX-81, deve recorrer a um programa tradutor compacto, eficiente e rápido, escrito em código de máquina e disponível comercialmente.

precedidos do caractere **$**; números binários, do caractere %. Qualquer número que não seja precedido de algum caractere será interpretado como decimal. Qualquer palavra que não seja um comando Assembly será considerada como um rótulo (label). Evite usar palavras parecidas ou muito longas; números não são permitidos.

Para que o Assembler saiba onde parar, termine seu programa com **REM end**.

Uma vez colocado o programa Assembly a ser montado (o *programa fonte*) nas linhas **REM** iniciais, basta rodar o Assembler para se obter uma listagem dos códigos equivalentes na tela: o *programa objeto*. O Assembler coloca o programa objeto simultaneamente na memória e na tela.

Se a esta altura for cometido algum erro na digitação das linhas, será suficiente, para corrigi-lo, listar o Assembler, editar a linha **REM** correspondente e usar novamente **RUN**.

Montado o programa, o endereço final da rotina em código de máquina aparecerá na tela.

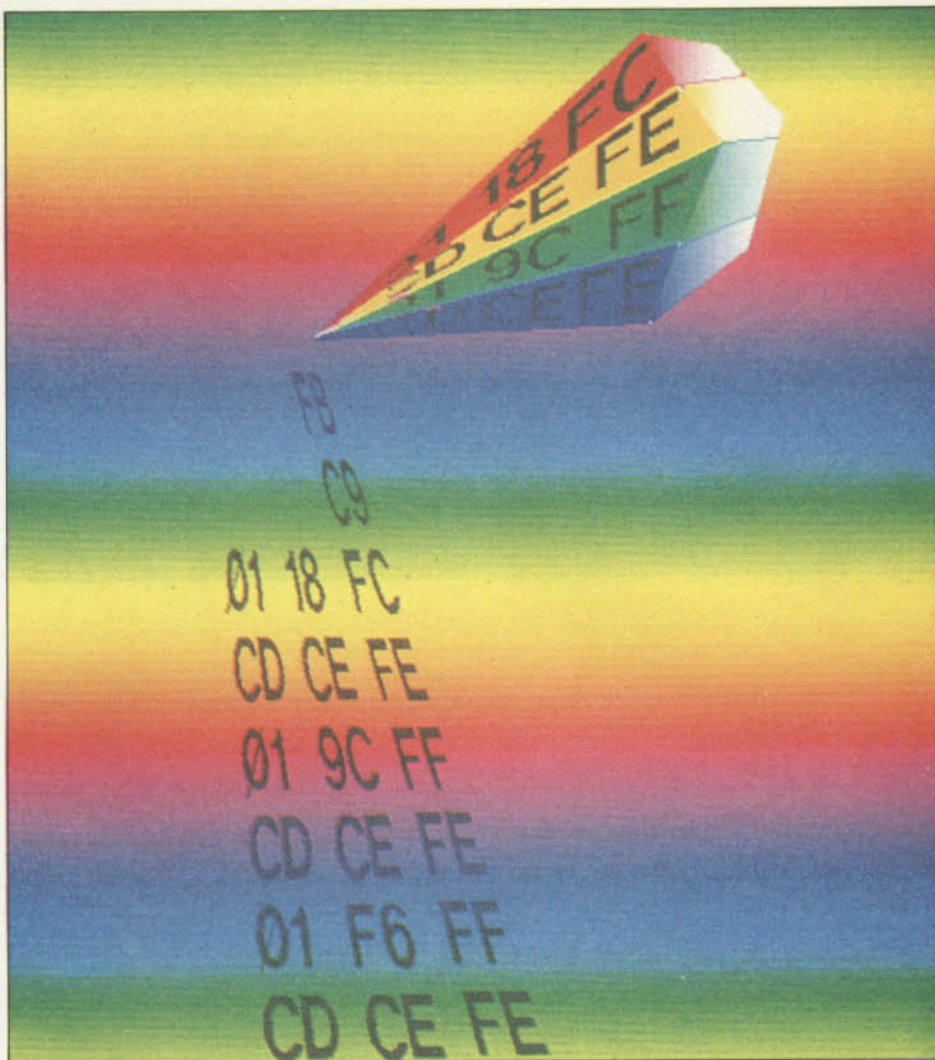
Para executar o programa em código usamos instruções do tipo **RANDOM USR**.

Se quisermos gravar o programa objeto, devemos digitar:

SAVE "nome" **CODE** endereço inicial, número de bytes.

COMO DETECTAR ERROS EM PROGRAMAS LONGOS

Sempre que a mensagem "**OUT OF DATA**" aparecer na tela, verifique se está faltando alguma coisa nas linhas **DATA** (lembre-se de que a ausência de uma simples vírgula pode deitar a perder todo o programa). Um dos erros mais comuns na digitação de longos programas consiste justamente na omissão de trechos de linhas **DATA**. Por isso, caso seu Assembler não funcione ao ser rodado pela primeira vez, não se desespere: ele certamente "empacou" devido a erros de digitação.

Esses errros podem ser detectados por meio de um programa de rastreamento, que escreve na tela o número da linha BASIC que está sendo executada. Um programa de rastreamento completo será apresentado mais adiante.

"Nome" é o nome do programa objeto e deve estar entre aspas. **CODE** informa ao computador que está sendo gravado um programa em código de máquina e não em BASIC. O endereço inicial é a origem do programa na memória. Podemos calcular o número de bytes subtraindo a origem do endereço final e somando 1.

O Assembler e o programa fonte podem ser gravados juntos, como um programa BASIC, usando-se **SAVE** da maneira habitual.

UM TESTE

Para testar seu programa, entre o programa de deslocamento horizontal da tela para a direita, que se encontra na página 215. Uma vez traduzido para código, esse programa ficará assim:

```
11 FF 57 21 FE 57 06 C0 C5 1A
01 1F 00 ED B8 12 2B 1B C1 10
F3 C9
```

Note que devemos usar letras minúsculas para digitar o programa fonte. Nosso Assembler não reconhece comandos em maiúsculas.

Não apresentaremos um Assembler para o ZX-81, pois é muito difícil escrever um programa desse tipo em BASIC para esse micro. Se você tem um ZX-81 e está interessado em código de máquina, sugerimos a compra de um Assembler gravado em fita cassete e disponível em lojas especializadas.

Caso já disponha de algum, teste o programa de deslocamento da tela para a direita apresentado na página 217 que, uma vez traduzido para código de máquina, ficará assim:

```
2A 0C 40 11 16 03 19 54 5D 13
06 18 C5 1A 01 1F 00 ED B8 12
2B 1B 1B C1 10 F1 C9
```

UM PROFESSOR DE DATILOGRAFIA

- PARA QUE APRENDER DATILOGRAFIA?
- AS TECLAS DE APOIO
- MELHORE A VELOCIDADE E A PRECISÃO NA DATILOGRAFIA

Aspiração de todo programador que se preza, a eficiência no processamento de textos e na digitação de programas tem um pré-requisito fundamental: saber datilografia.

O futuro nos reserva, certamente, muitas surpresas no campo das relações entre o homem e a máquina. Uma delas pode ser a abertura de novos canais de comunicação entre o computador e o usuário — como, por exemplo, a voz humana. Atualmente, alguns desses sistemas de controle mais sofisticados já vêm sendo aplicados em máquinas industriais e comerciais a um preço relativamente baixo. A maioria dos usuários, porém, vai ter que esperar muito tempo até ter acesso a essas inovações. Enquanto isso, o velho teclado continuará a ser o dispositivo básico de entrada de um computador.

Na verdade, o teclado é um instrumento de eficiência razoável para se transmitir instruções ao computador. Sua utilização, no entanto, consome muito tempo, especialmente quando o operador não é um exímio datilógrafo. Essa perda de tempo é visível quando tentamos copiar uma listagem e somos obrigados a olhar do teclado para a tela e desta para o papel, só para nos assegurarmos de que os dados estão sendo introduzidos corretamente.

Assim, por que não utilizar o computador no aprimoramento do modo de datilografar? Fazendo isso, você economizará tempo e evitará dores de cabeça, quando quiser escrever programas ou explorar o potencial de um computador como processador de textos. O programa que se segue ajudará a se familiarizar com o teclado e aumentar a velocidade e a precisão na datilografia.

O Sinclair Spectrum apresenta a desvantagem de ter um teclado de difícil manipulação e grande velocidade. A introdução de programas, contudo, exige apenas um toque de tecla para cada comando em BASIC. Note que não estamos apresentando uma versão do programa para o Sinclair ZX-81. Isso se deve ao fato de que sua execução se tornaria muito lenta, perdendo eficácia como forma de treinamento.

O PROGRAMA

Do mesmo modo que qualquer curso padrão de datilografia, nosso programa tem por objetivo desenvolver metodicamente a destreza do iniciante. Esse programa, no entanto, não é nenhum processo mágico e, como qualquer outra coisa que se queira aprender, exige dedicação.

Para efeitos didáticos, ele foi dividido em etapas mais ou menos fáceis, com seções extras de programação acrescentadas a cada uma das fases. Dessa maneira, é possível avançar através de misseis cada vez mais difíceis. Outra virtude de nosso curso consiste em apresentar, ao longo de seu desenvolvimento, uma atualização constante da velocidade e da precisão atingidas pelo iniciante.

Se você é usuário de um Apple II, ficará sem o recurso de cronometragem, devido à falta de um "clock" acessível nesse micro. Talvez a cronometragem

manual o auxilie, nos níveis 4 e 5, a controlar seus progressos na datilografia.

CONHEÇA AS TECLAS DE APOIO

Ser um exímio datilógrafo significa ser capaz de pressionar as teclas corretas sem precisar olhar para elas sempre que se quer localizá-las. Evidentemente, isso só poderá ser feito quando se sabe previamente onde se encontram as letras no teclado. É preciso, portanto, "educar" os dedos para que eles encontrem automaticamente as teclas desejadas. O primeiro passo para isso consiste em posicionar os dedos sempre na mesma posição sobre o teclado. Datilógrafos profissionais fazem isso assegurando-se de que os dedos das duas mãos repousem sempre sobre as mesmas teclas, as denominadas *teclas de apoio*. Estas são as teclas da fileira do meio do teclado, começando pelas letras A, S, D, F... Portanto, estas serão as teclas que devemos memorizar em primeiro lugar.

Uma vez dominada a localização dessas teclas básicas, o restante do teclado pode ser alcançado movendo-se cada dedo para cima ou para baixo.

Para obter o máximo rendimento no curso será preciso ater-se a essa disposição, sem nunca tentar "enganar" o computador. Por outro lado, os que insistem em continuar "catando milho" devem também procurar obter um gradativo aprimoramento de sua destreza.

Ao introduzirmos o programa — tarefa que deverá tornar-se cada vez mais fácil à medida que o curso prosseguir — devemos ficar com as mãos pousadas sobre o teclado, aguardando as instruções que aparecerem na tela e que serão explicadas a seguir.

O dedo mínimo da mão esquerda deve repousar sobre a tecla A, o anular sobre o S, assim por diante, terminando com o dedo indicador sobre a tecla F. O indicador direito, por sua vez, deve estar na tecla J, o dedo médio na tecla K, e assim por diante. Os polegares (exceto no caso do Spectrum) devem posicionar-se sobre a barra de espacejamento. As letras G e H, portanto, não são cobertas por nenhum dedo. Quando é necessário acioná-las, o G pode ser alcançado com o dedo indicador esquerdo, e o H com o indicador direito.

Parece complicado, mas não se preocupe: a primeira parte do programa se destina justamente a familiarizá-lo com as letras do teclado.

COMO FUNCIONA O PROGRAMA

Ao se executar o programa, a tela mostrará cinco opções, que correspon-

dem a níveis de dificuldade crescente. Você deve escolher uma entre elas. Se você está começando a aprender, precisa estudar primeiro a lição 1 para, só depois de dominar seu conteúdo, passar para a lição 2, e assim por diante.

dor de cada mão), voltando-se às teclas de apoio logo em seguida. Utilize o dedo mínimo direito para as teclas ; e Ç.

O asterisco continuará no mesmo lugar, caso seja cometido algum engano. Uma tecla batida corretamente será anunciada por um bip, e por um som grave se houver um erro ou se uma letra for omitida. (No Apple ocorrerá o oposto.)

Ao fim de cada exercício, a tecla mostrará o tempo gasto e o número de erros cometidos.

NÍVEL 3

Este nível também apresenta letras aleatoriamente, uma por vez. Agora, porém, a tela não indicará sua posição no teclado. Em vez disso, elas serão mostradas individualmente no centro da tela. Mais uma vez, as letras serão em número de vinte.

NÍVEL 4

Finalmente, pode-se datilografar algumas palavras. As expressões deste

LEVEL 3.

DIGITE A LETRA INDICADA NA TELA.

H

LEVEL 4...

DIGITE A PALAVRA INDICADA

FADA

NÍVEL 1

A tela apresentará as teclas de apoio em uma linha, na mesma ordem em que estão no teclado. Ao mesmo tempo, será mostrado um asterisco que, deslocando-se acima das letras, percorrerá o teclado do A até o L no Spectrum e no CP400, até o ; (ponto e vírgula) no Apple, até o Ç no MSX. Para se acionar as teclas, deve-se tocá-las na seqüência indicada pelo asterisco e com o dedo correto (lembre de que o G e o H exigem um deslocamento do dedo indica-

NÍVEL 2

Uma vez familiarizado com a posição das letras no teclado, você pode passar para este nível, que é semelhante ao primeiro, exceto que agora o asterisco se deslocará aleatoriamente sobre as letras (não vale trapacear olhando a posição da letra do teclado). Novamente, o tempo que se levar para completar o exercício (serão apresentadas vinte letras aleatoriamente) e o número de erros cometidos serão apresentados no final.

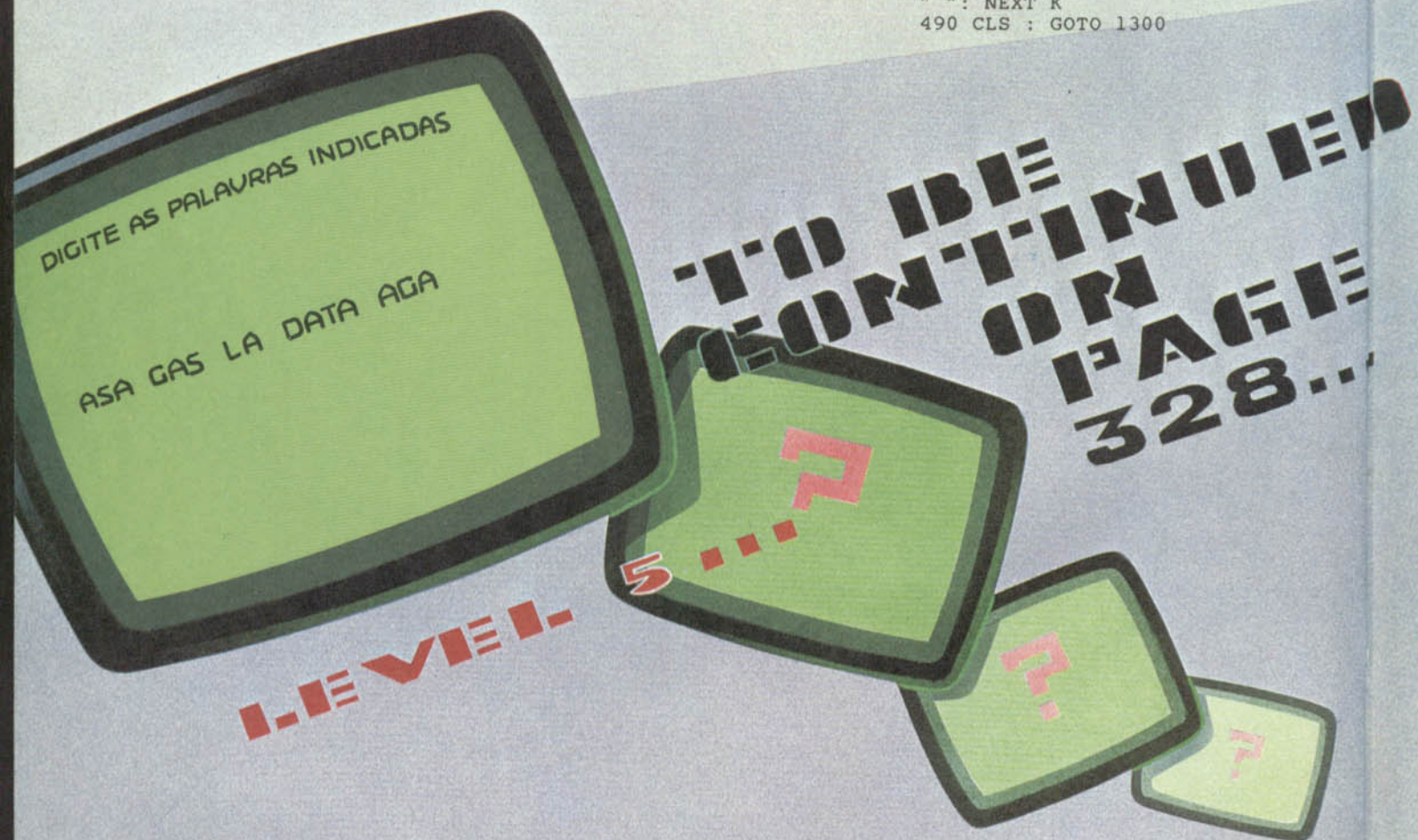
exercício utilizam apenas as letras das teclas de apoio: FADA, por exemplo. Ao contrário do nível anterior, serão apresentadas no centro da tela vinte palavras (e não letras) aleatoriamente e em sucessão, isto é, uma após a outra. À medida que as palavras forem sendo copiadas, um asterisco indicará cada uma de suas letras. E, mais uma vez, será apresentado o resultado do tempo gasto e da precisão. Não se deve trapacear olhando para o teclado; senão, não será pos-

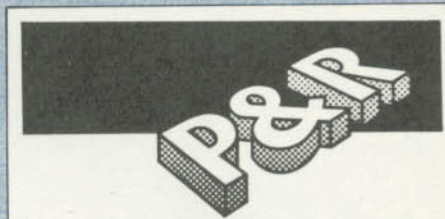
sível completar os testes mais difíceis, apresentados nas próximas lições.

NÍVEL 5

Desta vez será escrita uma sentença com cerca de seis palavras aleatórias. À medida que estas forem sendo digitadas, as letras aparecerão uma após a outra. Quando a linha toda tiver sido datilografada, será mostrado o desempenho do datilógrafo, assim como um valor correspondente à velocidade em termos de palavra por minuto. Ao se digitar as palavras que surgirem na tela, não se deve esquecer dos espaços em branco entre elas; esse espaço é obtido quando se toca na barra de espacejamento com um dos polegares. No Spectrum será preciso pressionar com o dedo mínimo a barra de espaços, situada numa posição um tanto incômoda.

Deve-se exercitar cada um dos níveis, até que se torne fácil executá-los. Nos próximos capítulos acrescentaremos letras e palavras novas.

```
 10 BORDER 1: PAPER 1: INK 7:
 CLS
 20 POKE 23658,8: LET ER=0
 30 LET S$="ASDFGHJKL"
100 PRINT  INVERSE 1;AT 8,3;"
PROFESSOR DE DATILOGRAFIA "
110 PRINT ''TAB 6;"SELECIONE N
IVEL (1 A 5)"
120 IF INKEY$="" THEN  GOTO
120
130 LET A$=INKEY$: IF A$<"1"
OR A$>"5" THEN  GOTO 120
140 SOUND .2,10
150 GOSUB VAL A$*100+100
160 GOTO 20
200 GOSUB 1000
210 FOR K=7 TO 23 STEP 2
220 PRINT AT 10,K;"*"
230 LET R$=S$((K-5)/2)
240 GOSUB 1100
250 IF C=0 THEN  GOTO 240
260 PRINT AT 10,K;" "
270 NEXT K
280 CLS : GOTO 1300
300 GOSUB 1000
310 FOR K=1 TO 20
320 LET RN=INT (RND*9)*2+1
330 PRINT AT 10,RN+6;"*": LET
R$=S$((RN+1)/2)
340 GOSUB 1100: IF C=0 THEN
GOTO 340
350 PRINT AT 10,RN+6;" "
360 NEXT K
370 CLS : GOTO 1300
400 CLS : PRINT "DIGITE A LETR
A INDICADA NA TELA"
410 FOR N=1 TO 100: NEXT N
420 POKE 23672,0: POKE 23673,0
430 FOR K=1 TO 20
440 LET RN=INT (RND*9)+1
450 PRINT AT 11,16;S$(RN)
460 LET R$=S$(RN)
470 GOSUB 1100: IF C=0 THEN
GOTO 470
480 PRINT  INVERSE 1;AT 11,16;
" ": NEXT K
490 CLS : GOTO 1300
```

```
500 CLS : PRINT "DIGITE A PALA
VRA INDICADA": POKE 23672,0:
POKE 23673,0
510 LET TL=0: FOR N=1 TO 20:
RESTORE : LET RN=INT (RND*24)+
1
520 FOR K=1 TO RN: READ T$:
NEXT K
530 PRINT AT 10,13;"      ":
PRINT AT 10,13;T$
540 FOR M=1 TO LEN T$: PRINT
AT 9,11+M;" *    "
550 LET R$=T$(M): GOSUB 1100
560 IF C=0 THEN  GOTO 550
570 NEXT M
580 LET TL=TL+LEN T$: NEXT N
590 CLS : PRINT AT 18,0;"PALAV
RAS POR MINUTO= ";INT (TL*500/
(PEEK 23672+256*PEEK 23673)+.5
): GOTO 1300
600 CLS : PRINT "DIGITE AS PAL
AVRAS INDICADAS": LET T$=""
610 FOR N=1 TO 6: RESTORE :
LET RN=INT (RND*24)+1: FOR K=1
TO RN: READ X$: NEXT K
620 LET T$=T$+X$+" "
630 NEXT N: LET T$=T$( TO LEN
T$-1)
640 POKE 23672,0: POKE 23673,0
: PRINT AT 10,0;T$
650 PRINT AT 12,0;: FOR M=1 TO
LEN T$
660 LET R$=T$(M): GOSUB 1100
670 IF C=0 THEN  GOTO 660
680 PRINT T$(M);: NEXT M
690 LET TL=LEN T$: GOTO 590
1000 CLS : PRINT "PRESSIONE A T
ECLA INDICADA PELO ASTERISCO"
1010 PRINT AT 12,7;"A S D F G H
 J K L"
1020 FOR K=1 TO 100: NEXT K: PO
KE 23672,0: POKE 23673,0
1030 RETURN
1100 PAUSE 0
1110 LET A$=INKEY$: IF A$<>R$ T
HEN  SOUND .2,-10: LET ER=ER+1:
 LET C=0: RETURN
1120 SOUND .05,20: LET C=1: RET
URN
1300 PRINT AT 19,0;"TEMPO= ";(P
EEK 23672+256*PEEK 23673)/50;"
SEGUNDOS": PRINT "NUMERO DE ERR
OS= ";ER
1310 PAUSE 100: RETURN
2000 DATA "FADA","GALA","SALA",
"DADA","ASA","LAKA","ALAGA","AL
ADA","LA","GAS","DALLAS","SAGA"
2010 DATA "AFAGA","FALHA","ADAG
A","HA","AGA","HAJA","HALL","GA
LHAS","FALA","FALHA","AJA","JAL
"
```

```
10 OB$="A S D F G H J K L"
20 CLS
30 DIM W$(28)
40 FOR K=1 TO 28
50 READ W$(K)
60 NEXT
70 PRINT @71,"QUAL O NIVEL DE"
80 PRINT @101,"DIFICULDADE (1-5
) ?"
90 PRINT @165,"DIGITE (0) PARA
SAIR"
100 A$=INKEY$:IF A$<"0" OR A$>"
5" THEN 100
110 A=VAL(A$):ON A+1 GOSUB 999,
200,300,400,600,800
120 ER=0
130 GOTO 70
200 GOSUB 1000
210 AP=1252
220 FOR K=1 TO 9
230 AP=AP+2
240 GOSUB 1100
250 NEXT
260 CLS:PRINT @448,"TEMPO=";TIM
ER/50;"SEGUNDOS"
270 PRINT"NUMERO DE ERROS=";ER;
280 RETURN
300 GOSUB 1000
310 FOR K=1 TO 20
320 AP=1252+2*RND(9)
330 GOSUB 1100
340 NEXT
350 CLS
360 PRINT @448,"TEMPO=";TIMER/5
0;"SEGUNDOS"
370 PRINT"NUMERO DE ERROS=";ER;
380 RETURN
400 CLS0:PRINT" DIGITE A TECLA
INDICADA NA TELA"
410 PRINT @206,"   ";:PRINT @23
8,"   ";:PRINT @270,"   ";
420 FOR K=1 TO 20
430 P$=MID$(OB$,2*RND(9)-1,1)
440 PRINT @239,P$;
450 A$=INKEY$:IF A$="" THEN 450
460 IF K=1 THEN TIMER=0
470 IF A$<>P$ THEN SOUND 5,1:ER
=ER+1:GOTO 450
480 POKE 1263,128
490 SOUND 200,1
500 NEXT
510 GOTO 350
600 CLS:PRINT" DIGITE A TECLA Q
UE APARECER"
610 T=0
620 FOR K=1 TO 10
630 P$=W$(RND(28))
640 T=T+LEN(P$)
650 PRINT @237,P$
660 P=1
670 POKE 1228+P,106
680 A$=INKEY$:IF A$="" THEN 680
690 IF K=1 THEN TIMER=0
700 IF A$<>MID$(P$,P,1) THEN ER
=ER+1:SOUND 5,1:GOTO 670
710 POKE 1228+P,96
720 SOUND 200,1:P=P+1:IF P<=LEN
(P$) THEN 670
730 NEXT
740 CLS
750 PRINT @416,USING"PALAVRAS P
OR MINUTO=###.##";T*500/TIMER
760 GOTO 360
800 CLS:P$="":FOR K=1 TO 6
810 P$=P$+W$(RND(28))+" "
820 NEXT
830 P$=LEFT$(P$,LEN(P$)-1)
840 PRINT" DIGITE ESTAS PALAVRA
S"
850 P=1:PRINT @224,P$
860 A$=INKEY$:IF A$="" THEN 860
870 TIMER=0:GOTO 890
```

Qual a velocidade de datilografia que devo tomar como meta para começar?

Na etapa inicial do curso, o mais correto é preocupar-se com a precisão e não com a velocidade. Depois que se consegue datilografar todas as palavras sem cometer nenhum erro, pode-se pensar em acelerar o ritmo de trabalho.

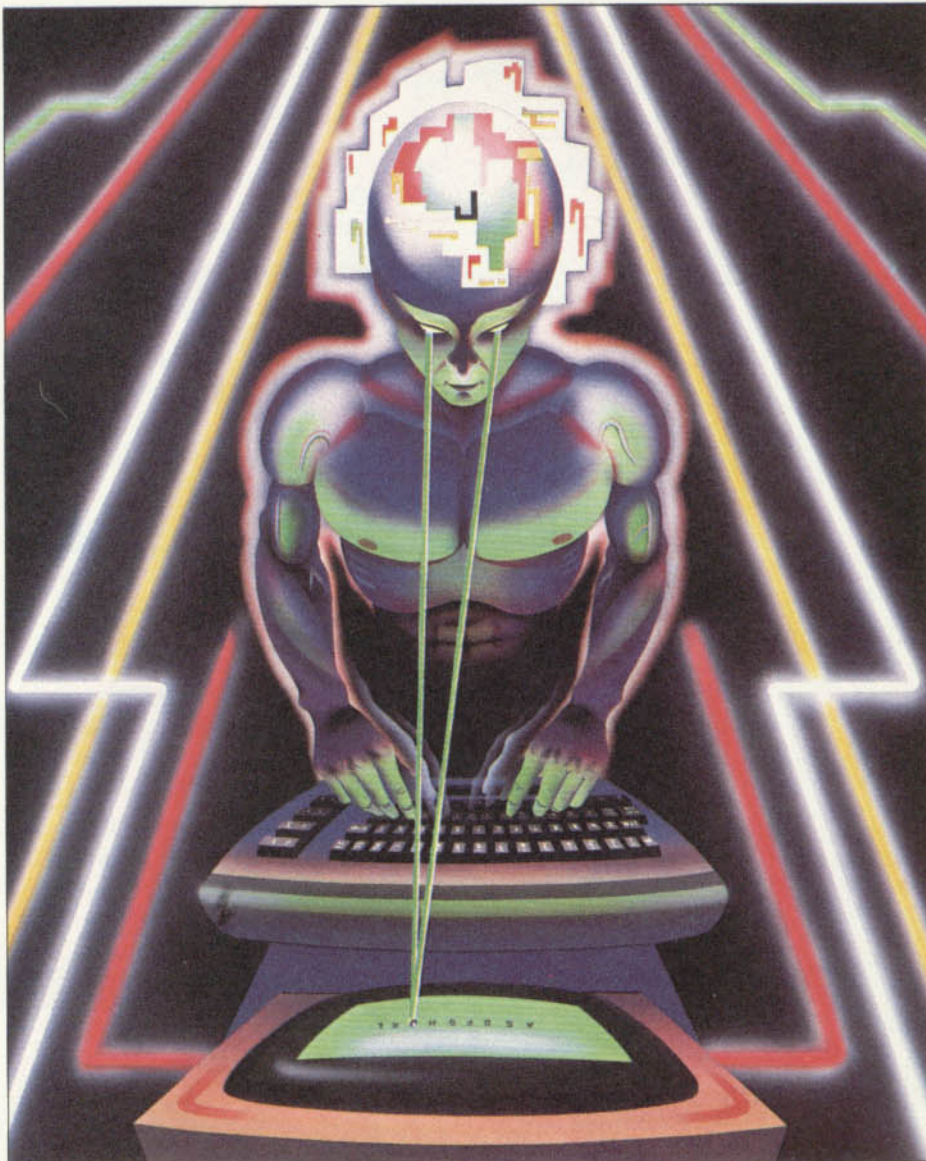
Deve-se praticar os Níveis 1, 2 e 3 até que se consiga datilografar vinte letras em doze ou treze segundos; nos Níveis 4 e 5, deve-se ter como meta cerca de quinze palavras ou mais por minuto.

```
880 A$=INKEY$:IF A$="" THEN 880
890 IF A$<>MID$(P$,P,1) THEN ER
=ER+1:SOUND 5,1:GOTO 880
900 POKE 1311+P,PEEK(1247+P)
910 SOUND 200,1:P=P+1:IF P<=LEN
(P$) THEN 880
920 T=LEN(P$):GOTO 740
999 CLS:END
1000 CLS:PRINT"PRESSIONE A TECL
A INDICADA PELO ASTERISCO"
1010 PRINT @262,OB$
1020 FOR K=1 TO 1000:NEXT
1030 RETURN
1100 POKE AP,106
1110 A$=INKEY$:IF A$="" THEN 11
10
1120 IF K=1 THEN TIMER=0
1125 PRINT@14*32,ASC(A$);
1126 PRINT@15*32,PEEK(AP+32);
1130 IF (ASC(A$))<>PEEK(AP+32)T
HEN SOUND 5,1:ER=ER+1:GOTO 1110
1140 SOUND 200,1
1150 POKE AP,96
1160 RETURN
9000 DATA FADA,GALA,SALA,DADA,A
SA,LAKA,ALAGA,ALADA,LA,GAS,DALL
AS,SAGA,GALHADA,FALA
9010 DATA AFAGA,FALHA,ADAGA,HA,
AGA,HAJA,JA,HALL,KA,FA,AJA,DAS,
GALHAS,JAL
```

```
5 R=RND(-TIME)
10 OB$="A S D F G H J K L Ç":Z$
=CHR$(219):S$="L10 O2 G"
20 SCREEN1:CLS
30 DIM W$(28)
40 FOR K=1 TO 28
50 READ W$(K)
60 NEXT
70 LOCATE 7,4:PRINT"Qual nível
de"
80 LOCATE 5,6:PRINT"dificuldade
```

```
? (1-5)"
90 LOCATE 3,9:PRINT"Tecle <0> p
ara terminar."
100 A$=INKEY$:IF A$<"0" OR A$>"
5" THEN 100
110 ON VAL(A$)+1 GOSUB 999,200,
300,400,600,800
120 ER=0
130 GOTO 70
200 GOSUB 1000
210 AP=357
220 FOR K=1 TO 10
230 AP=AP+2
240 GOSUB 1100
250 NEXT
260 CLS
270 LOCATE 5,12:PRINT"Tempo =";
INT(TIME/60);"segundos."
280 LOCATE 6:PRINT"Número de er
ros =";ER;
290 RETURN
300 GOSUB 1000
310 FOR K=1 TO 20
320 AP=359+INT(RND(1)*10)*2
330 GOSUB 1100
340 NEXT
350 GOTO 260
400 CLS:PRINT"Digite a letra mo
strada       na tela:"
410 LOCATE 11,9:PRINTZ$;Z$;Z$;Z
$:LOCATE 11:PRINTZ$;"  ";Z$:LOC
ATE 11:PRINTZ$;"  ";Z$:LOCATE11
:PRINTZ$;Z$;Z$;Z$
420 FOR K=1 TO 20
430 P$=MID$(OB$,INT(RND(1)*10)*
2+1,1)
440 LOCATE 13,11:PRINTP$;
450 A$=INKEY$:IFA$=""THEN450
460 IF K=1 THEN TIME=0
470 IF A$<>P$ THEN PLAY S$:ER=E
R+1:GOTO450
480 PRINTCHR$(8);CHR$(32);
490 BEEP
500 NEXT:GOTO 260
600 CLS:PRINT"Digite a palavra
que                 aparecer:"
610 TL=0
620 FOR K=1 TO 10
630 P$=W$(INT(RND(1)*28)+1)
640 T=LEN(P$):TL=TL+T:L=INT((30
-T)/2)
645 LOCATE 0,11:PRINTSPACE$(30)
;
650 LOCATE L,11:PRINTP$
655 FOR J=1 TO T
660 LOCATE L,10:PRINTCHR$(205);
670 A$=INKEY$:IFA$=""THEN670
680 IF K=1 THEN TIME=0
690 IF A$<>MID$(P$,J,1)THEN PLA
Y S$:ER=ER+1:GOTO 670
700 LOCATE L,10:PRINTCHR$(32);
710 BEEP:L=L+1:NEXT
720 NEXT
730 CLS:LOCATE 2,16:PRINT "Pala
vras por minuto:";:PRINT USING
"###.##";TL*600/TIME
740 GOTO 270
800 CLS:P$=""
810 FOR K=1 TO 5:P$=P$+W$(INT(R
ND(1)*28)+1)+CHR$(32):NEXT
820 P$=LEFT$(P$,LEN(P$)-1)
830 PRINT"Digite estas palavras
:"
840 P=1:LOCATE 0,11:PRINTP$
850 A$=INKEY$:IFA$=""THEN850
860 TIME=0:GOTO 880
870 A$=INKEY$:IFA$=""THEN870
880 IF A$<>MID$(P$,P,1)THEN PLA
Y S$:ER=ER+1:GOTO 870
890 LOCATE-1+P,14:PRINTMID$(P$,
P,1)
900 BEEP:P=P+1:IF P<=LEN(P$) TH
EN 870
910 TL=LEN(P$):GOTO 730
999 SCREEN0:END
1000 CLS:PRINT"Digite o caracte
r indicado      pela seta:"
1010 LOCATE 5,12:PRINTOB$
1020 FOR K=1 TO 1000:NEXT
1030 RETURN
1100 VPOKE BASE(5)+AP,205
1110 A$=INKEY$:IF A$=""THEN 111
0
1120 IF K=1 THEN TIME=0
1130 IF ASC(A$)<>VPEEK (BASE(5)
+32+AP)THEN PLAY S$:ER=ER+1:GOT
O 1110
1140 BEEP
1150 VBOKE BASE(5)+AP,0
```

```
1160 RETURN
9000 DATA FADA,SALA,ADAGA,DADA,
FAÇA,FA,GALA,HAJA,AGA,HALL,ALAD
A,ASA,GAS,FALHA
9010 DATA FALA,LAÇA,ALAGA,LAKA,
SAGA,JA,AJA,AFAGA,GALHADA,DALLA
S,LA,HA,ASSA,LAJA
```

```
10 OB$ = "A  S  D  F  G  H  J
K  L  ;"
20  HOME
30  DIM W$(28)
40  FOR K = 1 TO 28: READ W$(K)
: NEXT
50  HTAB 14: VTAB 5: PRINT "Qua
l nivel de"
60  PRINT : HTAB 12: PRINT "dif
iculdade? (1-5)"
70  HTAB 9: VTAB 11: PRINT "Tec
le <0> para terminar."
80  GET A$: IF A$ < "0" OR A$ >
 "5" THEN 80
90  ON  VAL (A$) + 1 GOSUB 999,
200,300,400,500,600
100 ER = 0
110  GOTO 50
200  GOSUB 1000
210 AP = 2
220  FOR K = 1 TO 10:AP = AP +
3: GOSUB 1100: NEXT
230  HOME
240  VTAB 15: HTAB 12: PRINT "N
umero de erros = ";ER;
250  RETURN
300  GOSUB 1000
310  FOR K = 1 TO 20
320 AP = 2 +  INT ( RND (1) * 1
0 + 1) * 3
330  GOSUB 1100: NEXT
340  GOTO 230
400  HOME : PRINT "Digite a let
ra mostrada na tela:"
410  VTAB 9: HTAB 18: INVERSE :
 PRINT "   ": HTAB 18: PRINT "
  ": HTAB 18: PRINT "   "
420  FOR K = 1 TO 20:P$ =  MID$
 (OB$, INT ( RND (1) * 10 + 1)
* 3 - 2,1)
430  VTAB 10: HTAB 19: PRINT P$
; CHR$ (8);
440  GET A$: IF A$ <  > P$ THEN
  PRINT  CHR$ (7);:ER = ER + 1:
 GOTO 440
450 A =  PEEK ( - 16336):A =  P
EEK ( - 16336): NEXT : NORMAL :
 GOTO 230
500  HOME : PRINT "Digite a pal
avra que aparecer:"
510 TL = 0
520  FOR K = 1 TO 10
530 P$ = W$( INT ( RND (1) * 28
) + 1)
540 T =  LEN (P$):L =  INT ((40
 - T) / 2)
550  PRINT : VTAB 11: CALL  - 9
58: HTAB L: PRINT P$
560  FOR J = 0 TO T - 1: VTAB 1
0: HTAB J + L: PRINT "*"; CHR$
(8);
570  GET A$: IF A$ <  >  MID$ (
P$,J + 1,1) THEN  PRINT  CHR$ (
```

```
7);:ER = ER + 1: GOTO 570
580 A =  PEEK ( - 16336):A =  P
EEK ( - 16336): PRINT  CHR$ (32
);: NEXT
585  NEXT
590  GOTO 230
600  HOME :P$ = ""
610  FOR K = 1 TO 6:P$ = P$ + W
$( INT ( RND (1) * 28) + 1) +
CHR$ (32): NEXT
620 P$ =  LEFT$ (P$, LEN (P$) -
 1)
630  PRINT "Digite estas palavr
as:"
640 P = 1: VTAB 11: HTAB 4: PRI
NT P$
650  GET A$: IF A$ <  >  MID$ (
P$,P,1) THEN  PRINT  CHR$ (7);:
ER = ER + 1: GOTO 650
660  VTAB 15: HTAB P + 3: PRINT
  MID$ (P$,P,1)
670 A =  PEEK ( - 16336):A =  P
EEK ( - 16336):P = P + 1: IF P
<  =  LEN (P$) THEN 650
680  GOTO 230
999  HOME : END
1000  HOME : PRINT "Digite o ca
racter indicado pelo          a
sterisco:"
1010  VTAB 12: HTAB 5: PRINT OB
$
1020  FOR K = 1 TO 500: NEXT
1030  RETURN
1100  POKE 1319 + AP,170
1110  GET A$: IF  ASC (A$) <  >
  PEEK (1319 + AP + 128) - 128
THEN  PRINT  CHR$ (7);:ER = ER
+ 1: GOTO 1110
1120 A =  PEEK ( - 16336):A =
PEEK ( - 16336): POKE 1319 + AP
,160
1130  RETURN
9000  DATA  FALA,AGA,AFAGA,GAS,
SALA,DADA,HAJA,HALL,LAKA,ASA,AL
ADA,ASDFG,SAGA,KAKA
9010  DATA  AJA,FA,ALAGA,JA,AGA
,HA,GALHADA,FALHA,GALA,FADA,DAL
LAS,ASSADA,LALA,;LKJH
```

CÓDIGO DE CONTROLE

Códigos de controle podem ser usados no lugar de muitos comandos em BASIC. Entrados diretamente por meio do teclado em certos micros, eles proporcionam uma ação imediata.

Os códigos de controle correspondem, na maioria dos micros, aos códigos de 0 a 31 no sistema ASCII, e servem para realizar uma série de funções relacionadas principalmente com o controle do vídeo.

Caso você não saiba ainda, ASCII é um sistema de codificação padronizado mundialmente e utilizado por quase todos os computadores. ASCII significa *American Standard Code for Information Interchange* (Código Padrão Americano para Intercâmbio de Informação), e se pronuncia *ásqui*.

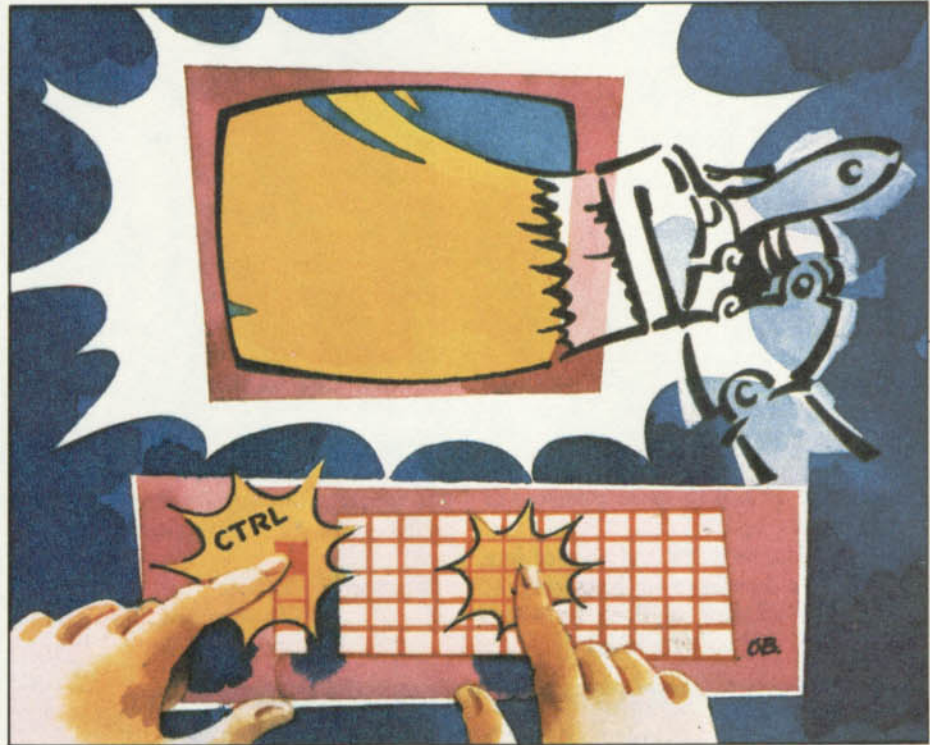
Originalmente, os códigos de controle do ASCII correspondiam às funções básicas de transmissão de um terminal ligado ao computador: início de transmissão, sinal de atenção, fim de transmissão, acionamento da campainha do terminal, etc.

Entretanto, cada fabricante de microcomputadores aproveitou a faixa de códigos de 0 a 31 para implementar funções diferentes. Com poucas exceções, portanto, a maioria dos códigos de controle não é padronizada, como acontece com os códigos correspondentes a caracteres comuns (letras, números, sinais matemáticos e de pontuação, etc.).

O ACESSO AOS CÓDIGOS DE CONTROLE

Existem duas técnicas básicas de utilização de códigos de controle. Uma delas — mais universal — pode ser aplicada a qualquer microcomputador que disponha da função **CHRS** no interpretador BASIC. A outra é aplicável apenas a alguns computadores: é a entrada direta dos códigos de controle por intermédio do teclado. A tecla rotulada **CONTROL** (ou ainda **CTRL**), existente em muitos micros, deve ser pressionada em combinação com outras teclas, para introduzir o código de controle numa linha de comando ou de programa.

A função **CHRS** (abreviatura de *character*) permite converter um código numérico inteiro, situado entre 0 e 255, ao seu caractere correspondente. Por exemplo, **PRINT CHRS(65)** escreverá na tela a letra A (maiúscula).

A função **CHRS** funciona também com os caracteres de controle. Dessa forma, se acionarmos um comando **PRINT** associado a um ou mais caracteres de controle, poderemos realizar uma série de tarefas básicas (até mesmo algumas que normalmente podem não ser acessíveis por intermédio de comandos BASIC). A seguir, damos alguns exemplos para o TRS-80 (os outros computadores serão abordados na continuação deste capítulo).

Você já conhece a instrução **CLS**, que serve para limpar a tela e retornar o cursor à posição superior esquerda do vídeo. Não seria interessante se pudéssemos apagar a tela apenas de um determinado ponto para baixo ou até o final de uma linha? Efeitos desse tipo são facilmente obtidos por meio da combinação do comando **PRINT CHRS(n)** com os seguintes códigos:

- 8 - recua e apaga o último caractere na tela;
- 10 - avança o cursor para o começo da linha seguinte;
- 14 - liga o cursor;
- 15 - desliga o cursor;
- 23 - converte a tela para a largura de 32 caracteres;
- 28 - retorna o cursor à posição 0,0, sem apagar a tela;
- 29 - move o cursor para o começo da linha;
- 30 - apaga da posição do cursor até o final da linha;
- 31 - apaga o espaço que vai da posição do cursor até o final da tela.

Uma seqüência de caracteres de controle pode ser colocada também dentro de uma cadeia de caracteres, por meio da função **STRING$**. Por exemplo: **PRINT STRING$(5,10)** provoca o avanço do cursor cinco linhas para baixo. Esse comando corresponde à ação de pressionar a tecla com a flecha para baixo cinco vezes seguidas, com a vantagem de poder ser colocado dentro de um programa (corresponde a cinco comandos **PRINT** em branco).

OS COMANDOS PEEK E POKE

- O EXAME DA MEMÓRIA
- ESCREVA NA TELA COM O POKE
- CARACTERES NO SPECTRUM
- CORES NO TRS-COLOR
- COMO CONTROLAR O APPLE II

Os comandos PEEK e POKE permitem examinar e alterar valores diretamente na memória do computador. E você pode usá-los sem dificuldade, incorporados a um programa BASIC.

Muitas vezes, usamos o computador sem nenhuma necessidade de saber como sua memória funciona. Quando escrevemos, por exemplo, **LET A = 67**, o computador por si só seleciona posições de memória livres, chama estas posições de "A" e nelas armazena o valor **67**. Se digitarmos, em seguida, **PRINT A**, o computador saberá exatamente onde encontrar o valor pedido — o processo é automático. Só é necessário indicar ao computador as posições de memória que devem ser usadas quando programamos em código de máquina.

Mas existe em BASIC uma maneira de examinar a memória do computador, para localizar os valores ali armazenados. É possível, também, armazenar valores apropriados na memória, a fim de alterar o comportamento da máquina.

As ferramentas BASIC para isso são o **PEEK** e o **POKE**. **PEEK** permite que se examine o valor armazenado na memória e **POKE** possibilita o armazenamento de um determinado valor.

COMO FUNCIONAM O PEEK E O POKE

Os computadores TRS-80, TRS-Color, MSX, bem como os ZX-81 e Spectrum de 48K, possuem 65 536 posições de memória endereçáveis; portanto, 64K. O Apple e o TK-2000, em suas diversas versões, apresentam diferentes tamanhos de memória. Uma parte dessa memória é a ROM, ou seja, a Memória Apenas para Leitura. Os conteúdos da ROM são fixos; assim, embora possamos examiná-la usando **PEEK**, não podemos utilizar o **POKE** para armazenar novos valores ou alterar os que nela já existem. O restante da memória é a RAM — a Memória de Acesso Aleató-

rio, ou Memória para Leitura e Escrita, como também é conhecida —, na qual se armazenam as variáveis e os programas BASIC. Podemos usar **PEEK** para examinar e **POKE** para armazenar os valores ali contidos.

Este programa permite que você examine qualquer posição de memória ROM ou RAM do computador:

```
10 INPUT "ENDERECO...";A
20 LET N=PEEK(A)
30 PRINT "CONTEUDO...";N
40 GOTO 10
```

Entre com o número que desejar, de 0 a 65 535, e você verá o que existe na posição indicada. Pode ocorrer, porém, que ao usar **PEEK** para examinar determinadas áreas da memória, você receba como resultado um valor fictício e não o valor real lá existente. Os conteúdos da RAM dependerão sempre do que você estiver fazendo com o computador; apenas os conteúdos da ROM são fixos.

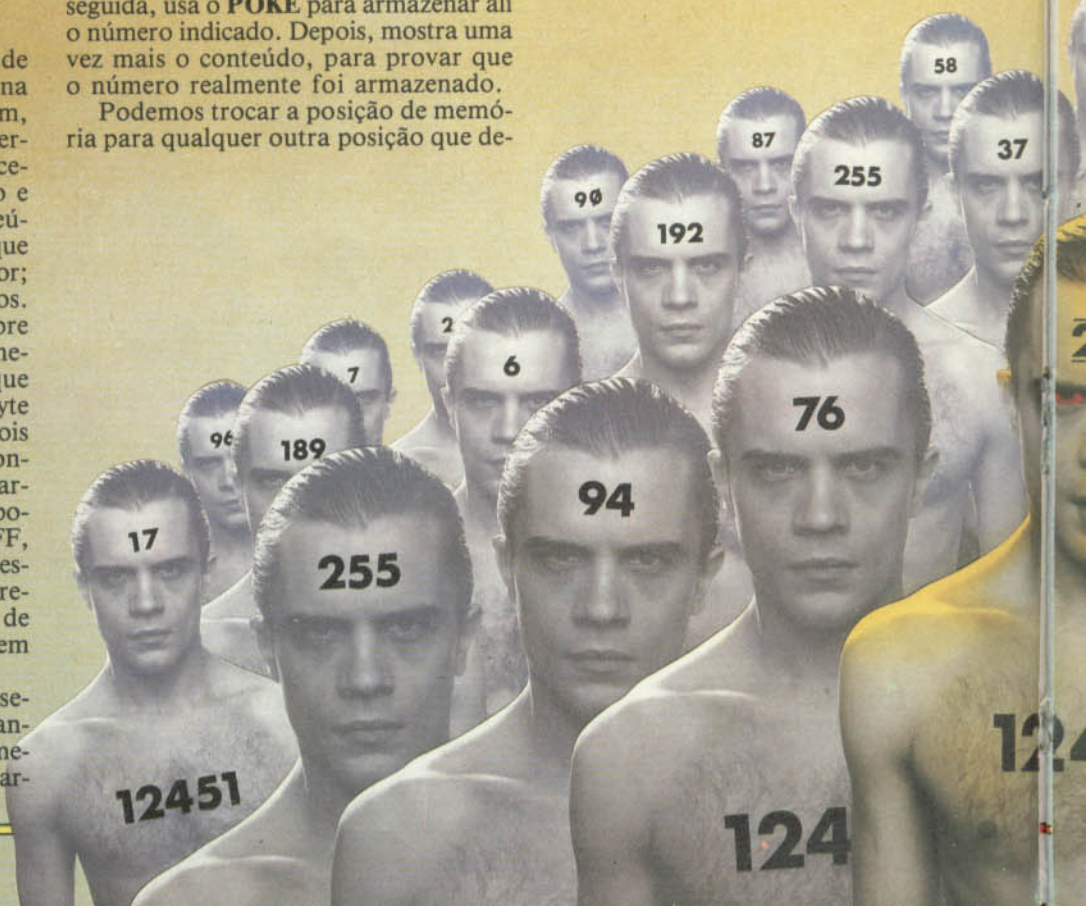
Repare que os números são sempre valores inteiros entre 0 e 255 e, em hexadecimal, de 0 a FF. Isso significa que todos constituem bytes simples (um byte contém um número hexadecimal de dois dígitos). Cada posição de memória contém um byte, não sendo possível guardar qualquer número maior em uma posição de memória. Se você somar 1 a FF, em hexadecimal, obterá 0100. Como este resultado inclui dois bytes, será preciso armazená-lo em duas posições de memória: 01 em uma posição e 00 em outra.

O programa que apresentamos a seguir permite que se empregue o comando **POKE** para guardar números na memória do computador. Neste estágio, armazene apenas bytes simples, ou seja, qualquer número entre 0 e 255.

```
10 PRINT"CONTEUDO...";PEEK(3000
)
20 INPUT"NUMERO....";N
30 POKE 30000,N
40 PRINT"NOVO CONTEUDO";PEEK(30
000)
45 PRINT
50 GOTO 20
```

O programa apresenta primeiro o conteúdo da posição de memória e, em seguida, usa o **POKE** para armazenar ali o número indicado. Depois, mostra uma vez mais o conteúdo, para provar que o número realmente foi armazenado.

Podemos trocar a posição de memória para qualquer outra posição que desejarmos. Repare que nada acontece se tentarmos usar **POKE** na ROM e não causaremos nenhum dano ao sistema se o fizermos. Utilizado em certas áreas da RAM, porém, o **POKE** pode provocar algum desarranjo, mas nada fatal — simplesmente desligue a máquina por um momento, para reiniciar a memória.

Tente agora utilizar o **POKE** para armazenar um número maior do que 255 e veja o que acontece. Nos computadores TRS-Color e Spectrum, você obterá uma mensagem de erro, já que só há lugar para um byte em cada posição de memória.

ESCREVA NA TELA COM O POKE

Examinando os mapas de memória apresentados no artigo da página 174, você verificará que uma porção da memória é dedicada à exibição em tela. Se usar o **POKE** para armazenar certos números nesta área, os caracteres realmente aparecerão na tela. Para ver determinado caractere, armazene códigos ASCII, empregando o **POKE**, nos computadores TRS-80, TRS-Color e TK-2000. No Apple, os valores ASCII produzirão caracteres piscantes; para obter caracteres normais, some 128 ao código ASCII. Nos micros da linha Sinclair e MSX, utiliza-se um método diferente do descrito abaixo. No TK-2000, a organização da tela não permite tal exibição. Tente as seguintes linhas:

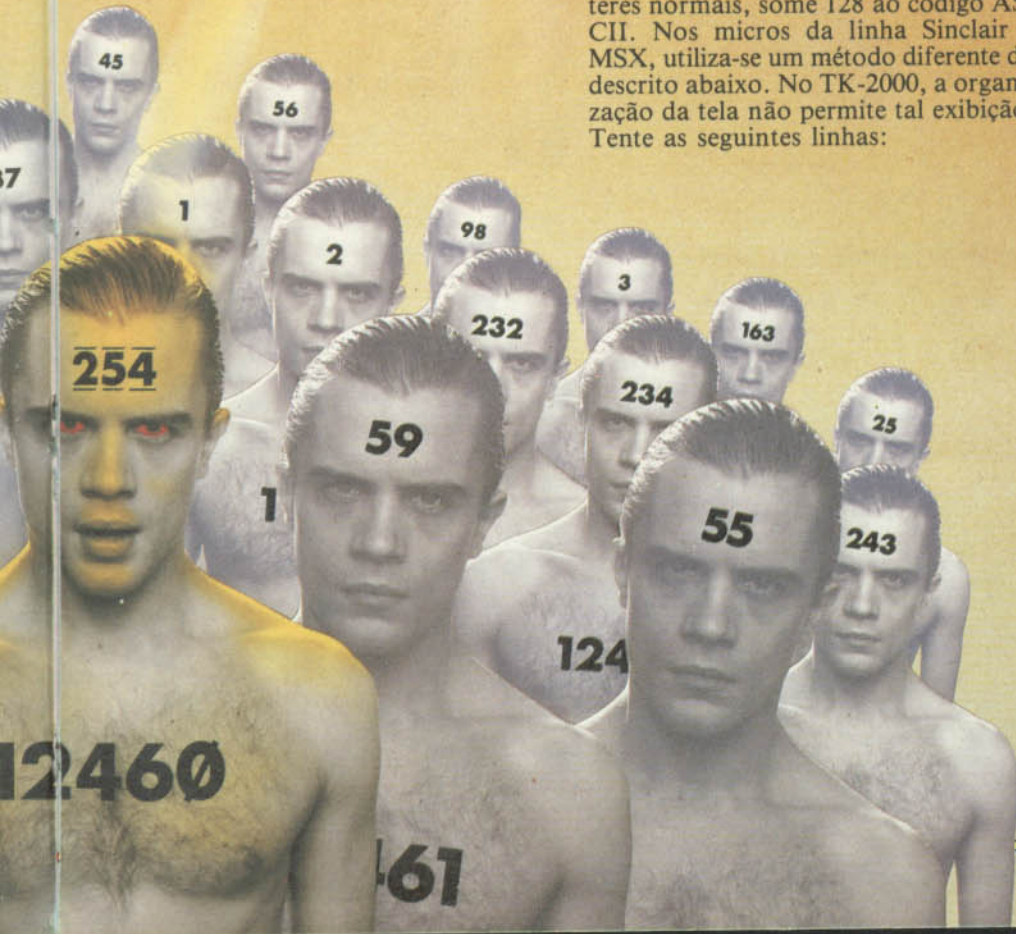
TABELA DE CÓDIGO ASCII

Cada caractere que o computador utiliza possui um código numérico, e muitos modelos adotam um código padrão denominado Código ASCII — iniciais de *American Standard Code for Information Interchange* (Código Padrão Americano para Intercâmbio de Informação).

Estes códigos são os números usados em **CHR$** e **ASC** (ou **CODE** no Spectrum). **CHR$** converte o número em um caractere, enquanto **ASC** ou **CODE** faz o inverso, convertendo um caractere em seu código numérico.

Assim, quando o computador guarda uma palavra, o que armazena é o código ASCII do caractere.

Tabela ASCII

Número de código	Caractere ASCII	Número de código	Caractere ASCII
32	space	62	>
33	!	63	?
34	''	64	@
35	#	65	A
36	$	66	B
37	%	67	C
38	&	68	D
39	'	69	E
40	(	70	F
41	)	71	G
42	★	72	H
43	+	73	I
44	,	74	J
45	—	75	K
46	•	76	L
47	/	77	M
48	0	78	N
49	1	79	O
50	2	80	P
51	3	81	Q
52	4	82	R
53	5	83	S
54	6	84	T
55	7	85	U
56	8	86	V
57	9	87	W
58	:	88	X
59	;	89	Y
60	<	90	Z
61	=		

```
POKE 15360,65
```

```
POKE 1024,65
```

```
HOME:POKE 1024,193
```

A letra A deverá aparecer no canto superior esquerdo da tela.

No TRS-Color, se a tela tiver rolado antes de uma tentativa de armazenar caracteres pelo **POKE**, eles não aparecerão em lugar nenhum. Assim, primeiramente limpe a tela, digitando **CLS** e, depois, <**RETURN**> ou <**ENTER**>.

Você deve estar se perguntando qual a vantagem de usar o **POKE** para colocar caracteres na tela, já que dispomos de comandos tão eficientes como o **PRINT A**, **PRINT TAB** ou **PRINT@**. De fato, o **PRINT** é normalmente o método mais fácil e conveniente para posicionar os caracteres. Mas há casos em que o emprego do **POKE** se mostra vantajoso. Por exemplo, se usarmos o **PRINT** para escrever na última posição da tela, esta sempre rolará; com o **POKE**, ao contrário, não haverá nenhum problema. Você pode imaginar o quanto esta característica é útil, sobretudo em jogos, quando queremos mover um caractere por toda a tela.

OS CARACTERES NO SPECTRUM

No Spectrum, as coisas são um pouco diferentes, pois não se pode usar o **POKE** para armazenar caracteres sobre a tela. Um caractere é feito de pontos de uma matriz de oito por oito. E cada linha deve ser armazenada pelo **POKE** separadamente, até se completar o caractere.

Felizmente, as formas dos caracteres são armazenadas na **ROM**; assim, pode-se usar o **PEEK** nesta porção da memória, a fim de examinar cada linha e, então, exibi-la na tela com o **POKE**.

O programa que apresentamos a seguir utiliza o comando **POKE** para exibir a letra A no canto superior esquerdo da tela.

```
10 LET dest=16384
20 FOR a=15880 TO 15887
30 POKE dest,PEEK a
40 LET dest=dest+256
50 NEXT a
```

A linha 10 define o endereço da primeira linha dos caracteres da tela. O **FOR...NEXT** permite o acesso à ROM, onde o caractere está armazenado e a linha 30 imprime a primeira linha deste caractere na tela. A linha 40 incrementa o endereço da tela em 256, dando acesso à próxima linha, logo abaixo da primeira.

O método funciona, mas, por ser um tanto lento, quase sempre é mais vantajoso usar o **PRINT AT** ou **PRINT TAB** no Spectrum.

O USO DO PEEK NA ROM E NA RAM

Quando executamos o primeiro programa que examina a memória do computador, obtivemos apenas números entre 0 e 255. Mas muitos desses números são, na realidade, códigos de letras em ASCII, alguns dos quais formam palavras ou mesmo sentenças completas. Se você não conhece os códigos ASCII, veja a tabela da página 263.

O programa seguinte lê toda a memória ROM do computador e converte os números em caracteres, antes de imprimi-los na tela:

```
10 FOR A=0 TO 16383
20 LET N=PEEK A
30 IF N>31 AND N<127 THEN PRINT
 CHR$N;
40 NEXT A
```

```
10 FOR A=&H8000 TO &HBFF
20 N=PEEK(A)
30 IF N>31 AND N<127 THEN PRINT
 CHR$(N);
40 NEXT A
```

```
10 FOR A=&H0 TO &H7FFE
20 N=PEEK(A)
30 IF N>31 AND N<127 THEN PRINT
 CHR$(N);
40 NEXT A
```

```
10  FOR A = 53248 TO 65535
20 N =  PEEK (A)
30  IF N > 31 AND N < 127 THEN
 PRINT  CHR$ (N);
40  NEXT A
```

As linhas 10 e 40 determinam a repetição do processo em toda a memória; a linha 20 examina cada posição, usando o **POKE**, e a linha 30 converte o número em um caractere e o imprime. Esta linha ainda limita o intervalo de números que serão convertidos, evitando que o computador imprima códigos de controle ou símbolos gráficos.

Para o TRS-80, use o programa do TRS-Color com a linha 10 do programa do Spectrum.

Pode-se também imprimir o conteúdo da RAM do mesmo modo, bastando trocar os endereços de memória da linha 10. Os endereços dados abaixo imprimem parte da área em que os programas BASIC estão armazenados, permitindo que você veja como o seu próprio programa está guardado na memória. Convém desligar o computador por um segundo antes de rodar este programa, para evitar que a RAM permaneça carregada com restos de outros programas. Aqui está a nova linha 10:

```
10 FOR A=23755 TO 65000
```

```
10 FOR A=16510 TO 30000
```

```
10 FOR A=&H1E00 TO &H1F00
```

Use **FOR 17385 TO 32767** para sistemas de 16K e os finais **49151** para 32K e **65535** para 48K.

```
10  FOR A = 2048 TO 32767
```

```
10 FOR A=&H8000 TO &HF37F
```

Experimente examinar outras áreas da RAM da mesma maneira.

APLICAÇÕES

Até aqui examinamos a memória do computador e usamos **POKE** para exibir caracteres na tela. Isso nos deu uma boa noção de como o **PEEK** e o **POKE** funcionam, mas não do quanto são úteis.

Para as aplicações mais importantes, precisaremos examinar posições específicas de memória. No Spectrum, a posição de memória 23609 controla o som que o teclado faz quando se pressiona uma tecla. Normalmente, ela contém 0, que provoca um pequeno clic; mas, com um número maior — que pode chegar a 255 —, obtém-se um longo bip. **POKE 23609,80** faz um ruído razoável.

As possibilidades de uso do **PEEK** e do **POKE** dependem principalmente do computador que você tem.

Retomando o exemplo anterior: os teclados dos demais computadores não fazem ruído quando uma tecla é pressionada; não têm posição de memória que controle o som do teclado. Assim, um efeito que requer cinco ou seis **PEEK** ou **POKE** em determinado computador pode ser obtido com uma palavra-chave BASIC em outra máquina.

O Apple faz um bom uso do **PEEK** e do **POKE**. O Spectrum, o TRS-80 e o TRS-Color utilizam-no ocasionalmente. No MSX o uso destes comandos é mais raro.

Aqui estão algumas aplicações para o seu computador.

Já vimos um **POKE** modificar o som do teclado. O **POKE** seguinte altera o espaço de tempo que uma tecla leva para iniciar a auto-repetição — o que é útil em jogos de velocidades, nos quais uma tecla precisa ser pressionada para mover um caractere. Neste caso e nos seguintes, **X** é uma variável que devemos ajustar por meio do comando.

```
POKE 23561,X
```

Normalmente, quando se liga o computador, **X** é ajustado para 35. Assim, use um número menor que 35 para diminuir o intervalo de espera e um número maior para obter um intervalo mais longo.

Pode-se também alterar a repetição automática com:

```
POKE 23562,X
```

Este tempo **X** é, em geral, igual a 5. Assim, use **X = 1** para uma auto-repetição rápida e **X = 25** para uma bem lenta.

CONTROLE DE TEMPO

O Spectrum não tem acesso ao cronômetro interno via teclado. Para utilizá-lo, precisaremos usar o **PEEK** na memória. O tempo é armazenado como três bytes em três posições consecutivas de memória.

```
PRINT(PEEK 23672+256*PEEK 23673
+65536*PEEK 23674)
```

... informará quantos cinqüenta avos de segundo o seu Spectrum está ligado desde o último **NEW**. Pressione **NEW** para reiniciar o cronômetro ou **POKE 0** em cada uma das três posições.

MAIS POKE

Apresentaremos aqui mais duas aplicações do **POKE**, que serão úteis quando você quiser escrever programas para outra pessoa. A primeira delas assegura que todas as letras digitadas apareçam em tipo maiúsculo.

```
POKE 23568,8
```

Esta instrução é útil quando se pretende que todos os dados de entrada sejam consistentes. Use **POKE 23658,0** para retornar ao normal.

Uma outra possibilidade é a alteração do cursor para qualquer caractere do teclado — ou mesmo qualquer palavra-chave, dependendo do número que foi usado no **POKE**:

```
10 FOR X=1 TO 255
20 POKE 23617,X
30 INPUT a$
40 NEXT X
```

A linha 30 faz um **INPUT a$** apenas para que o cursor apareça na tela. Tão logo entre algum dado via **INPUT**, o programa mostrará o cursor seguinte. Os mais interessantes são aqueles por volta de **X** = 200 até 230. Por exemplo, **X = 210** exibe **<CONTINUE>** como um cursor, e **X = 225** exibe um sinal de interrogação.

O **PEEK** e o **POKE** são também utilizados para a auto-repetição. O TRS-Color normalmente não tem teclado auto-repetitivo. Para mover objetos em jogos, por exemplo, é preciso manter a tecla pressionada, soltá-la muito rapidamente, voltar a pressioná-la e, assim, sucessivamente. Uma maneira de evitar esse cansativo procedimento é usar **PEEK** para verificar que tecla está sendo pressionada, o que nos permite deixar o dedo sobre a tecla pelo tempo que quisermos.

Esse método já foi empregado na seção de Programação de Jogos. Veja como funciona: quando pressionamos uma tecla, um código numérico especial é colocado em uma das seis posições de memória. O **PEEK** checa, então, estas posições, para verificar que tecla está sendo pressionada. O emprego deste recurso é muito simples. O programa seguinte, por exemplo, usa teclas de cursor para desenhar na tela:

```
10 PMODE 0,1
20 PCLS
30 SCREEN 1,1
40 X=127:Y=95
50 IF PEEK(341)=223 THEN Y=Y-2
60 IF PEEK(342)=223 THEN Y=Y+2
70 IF PEEK(343)=223 THEN X=X-2
80 IF PEEK(344)=223 THEN X=X+2
90 IF X<0 OR X>255 THEN X=-255*
(X<0)
100 IF Y<0 OR Y>191 THEN Y=-191
*(Y<0)
110 PSET (X,Y,5)
120 GOTO 50
```

A tabela da página 265 mostra o código numérico gerado para cada tecla e a posição de memória em que elas são armazenadas. Se, por exemplo, pressionássemos A, **PEEK (339)** seria igual a 267. Podemos verificar, assim, de onde vieram os números escritos após os **PEEK** nesse último programa.

TELA MULTICOLORIDA

O programa seguinte traça linhas de diferentes cores na tela. Colocando-as perto umas das outras, obteremos novos matizes. Você pode pressionar qualquer tecla durante a execução do programa, se quiser interromper o desenho e examinar o número para determinado matiz.

```
10 PMODE 3,1:SCREEN 1,0:PCLS3
20 FOR K=0 TO 255
30 POKE 178,K
40 LINE(78,46)-(178,146),PSET,B
F
50 IF INKEY$<>"" THEN 80
60 NEXT
70 GOTO 70
80 CLS:PRINT K
```

A posição de memória 178 controla a cor do primeiro plano. Esta varia, em geral, de 0 a 3 (4 cores), mas, se usarmos o **POKE** para armazenar um valor maior que **K = 3**, obteremos novas listras coloridas.

Utilizamos **PSET** na linha 40 porque estamos manipulando cores de primeiro plano. Para trabalhar as cores de plano de fundo, troque **PSET** por **PRESET** e substitua o valor 178 na linha 30 por 179.

Parando o programa, poderemos examinar o valor de **K** para determinado matiz e, assim, usar esta cor em outros programas gráficos.

COMO ALTERAR A VELOCIDADE

A possibilidade de acelerar o computador é interessante, sobretudo, para jogos — você pode fazê-los andar mais depressa se, por exemplo, o jogador atingir um determinado número de pontos.
Utilize o primeiro **POKE** para acelerar e o segundo para voltar ao normal.

```
POKE 65495,1
POKE 65494,1
```

Pode-se usar qualquer número em lugar do 1, pois o efeito será o mesmo.
Existe apenas um problema com estes **POKE** — como eles não funcionam em qualquer processador, corre-se o risco de que o programa prejudique o sistema. Mas não custa nada experimentá-los uma vez em seu computador.
E, agora, se você considera determinado jogo muito rápido, aqui está uma maneira de desacelerar a saída na tela.

```
POKE 359,60
```

Para voltar ao normal, use **POKE 359,67**, mas não no CP-400 Color, pois ele desativa o retorno automático ao modo texto.

No Apple é possível controlar a porção da tela que iremos utilizar. O endereço 32 contém a margem esquerda; 33, a largura máxima da linha; 34, o número de linhas da margem superior e 35, da margem inferior. Modificando os valores desses endereços com **POKE**, pode-se programar a janela de textos. É o que faz o programa seguinte.

```
5  HOME : VTAB 5: HTAB 11
10  POKE 32,10: POKE 33,20
20  POKE 34,4: POKE 35,20
30  PRINT "JANELA ";
40  GOTO 30
```

Outra utilidade destes comandos é possibilitar uma "varredura do teclado", verificando o valor contido na posição −16384 — sendo maior que 127, sabe-se que alguma tecla foi pressionada. Na realidade, esta posição de memória contém o código ASCII do caractere da tecla pressionada mais 128.
Sempre que fizermos esse tipo de "varredura", devemos colocar o valor 0 na posição −16368, para que uma nova verificação seja possível. O desrespeito a esta regra pode prejudicar seus programas. O programa que se segue exemplifica a utilização desse recurso.

```
10  HOME
20 A =  PEEK ( - 16384): POKE
- 16368,0
30  IF A > 127 THEN  PRINT  CHR
$ (A - 128);
40  GOTO 20
```

O programa funciona como uma máquina de escrever. Os caracteres correspondentes às teclas pressionadas são mostrados no vídeo.
O uso do **PEEK** também possibilita a produção de sons. Toda vez que verificamos o conteúdo do endereço −16336, um clic é produzido. Faça uma experiência, adicionando as linhas abaixo ao programa anterior.

```
30  IF A > 127 THEN  PRINT  CHR
$ (A - 128);: FOR I = 1 TO 10:X
 =  PEEK ( - 16336): NEXT
```

COMO ALTERAR A TELA

O Apple pode mostrar texto e gráficos de baixa e alta resolução em duas "páginas diferentes". Alguns **POKE** são capazes de modificar rapidamente as características da tela.

```
POKE -16304,0
```

Troca a tela de textos pela tela gráfica, sem limpar esta última.

```
POKE -16303,0
```

Troca a tela gráfica de qualquer tipo pela tela de textos, sem redefinição da janela de textos.

```
POKE -16302,0
```

Permite a exibição de gráficos em tela cheia, ou seja, impede o aparecimento de texto nas quatro linhas inferiores da tela gráfica.

```
POKE -16301,0
```

Troca a tela cheia por gráfico mais texto, ou seja, faz reaparecer as quatro linhas de texto na parte inferior da tela.

```
POKE -16300,0
```

Passa da página 2 para a página 1. Não limpa a tela nem move o cursor.

```
POKE -16299,0
```

Passa da página 1 para a página 2, da mesma maneira.

```
POKE -16298,0
```

Passa do modo gráfico de alta resolução para o modo texto, sem limpar a tela.

```
POKE -16297,0
```

Passa do modo texto para o modo gráfico de alta resolução.

Como o BASIC do MSX é muito poderoso, são poucos os comandos **PEEK**

PEEK e POKE de teclado do TRS-Color

Endereço / Número de código

Endereço	191	223	239	247	251	253	254
338	ENTER	8	∅	X	P	H	@
339	CLEAR	9	1	Y	Q	I	A
340		:	2	Z	R	J	B
341		;	3	↑	S	K	C
342		,	4	↓	T	L	D
343		—	5	←	U	M	E
344		•	6	→	V	N	F
345		/	7	SPACE	W	O	G

e **POKE** realmente úteis.

O modo como este computador organiza sua memória de vídeo impede que coloquemos caracteres diretamente na tela, usando **POKE**. Para isto, existe no BASIC do MSX o comando **VPOKE**, bem como o correspondente **VPEEK**. Os 16 kbytes da memória de vídeo só podem ser acessados por meio desses dois comandos.

O computador utiliza essa memória não só para mostrar caracteres e gráficos; parte dela armazena informações para o vídeo, tal como o padrão das letras do conjunto de caracteres.

O modo como isso é feito — ou seja, a organização da memória de imagem — varia conforme o tipo de tela escolhida pelo comando **SCREEN**.

Para que não tenhamos que decorar uma grande quantidade de endereços de memória, o MSX usa variáveis do sistema para guardar os endereços.

Vamos tentar entender esta organização. No modo texto de 40 colunas **(SCREEN 0)**, a variável **BASE (0)** contém o endereço inicial de uma porção da memória denominada tabela de nomes. Qualquer valor colocado em uma das 960 posições, a partir deste endereço, fará surgir na tela o caractere de código ASCII correspondente. Assim, o programa seguinte enche a tela com o caractere A, cujo código é 65.

```
10 SCREEN 0
20 FOR I=0 TO 959
30 VPOKE BASE(0)+I,65
40 NEXT
```

Note que, para colocarmos o caractere A na i-ésima posição da tela, devemos usar **VPOKE BASE (0) + I,65**.

No modo texto de 32 colunas, **BASE (5)** contém a tabela de nomes e existem 768 posições na tela. Assim, temos um programa análogo.

```
10 SCREEN 1
20 FOR I=0 TO 767
30 VPOKE BASE(5)+I,65
40 NEXT
```

O modo gráfico de alta resolução também possui uma tabela de nomes, cujo endereço inicial fica em **BASE (10)**, com 768 posições. O computador multiplica por 8 o número ali colocado por **VPOKE**. Em seguida usa o resultado dessa operação para obter o padrão e as cores daquela posição. O padrão dos pontos é armazenado em outra parte da memória de vídeo, cujo endereço inicial está em **BASE (12)**. Outra posição de memória é reservada para as cores dos pontos e seu início está em **BASE (11)**. Para entender melhor o processo, digite o seguinte programa:

```
10 SCREEN 2
20 FOR I=0 TO 767
30 VPOKE BASE(10)+I,0
40 NEXT
50 FOR I=0 TO 7
60 VPOKE BASE(11)+I,RND(1)*256
70 VPOKE BASE(12)+I,RND(1)*256
80 NEXT
90 GOTO 50
```

As linhas de 20 a 40 enchem a tabela de nomes com o número 0. Este número é multiplicado por 8, dando 0. Isso significa que o computador irá aos oito primeiros endereços a partir da posição 0 na tabela de padrões, para obter o padrão; e aos oito primeiros endereços a partir da posição 0 da tabela de cores, para obter a combinação de cores destas posições. Como ainda não há nada nas duas tabelas, nenhum desenho é feito a princípio.

As linhas de 50 a 80 preenchem as oito primeiras posições, a partir do endereço 0, da tabela de padrões com números aleatórios. Fazem o mesmo com os endereços da tabela de cores. Estes são justamente os locais onde o computador vai buscar a cor e o padrão de posições da tabela de nomes que tenham o número 0. Como a tabela de nomes está cheia de zeros, todas as posições do terço superior da tela assumirão a mesma cor e padrão simultaneamente. A linha 90 repete o processo.

Você deve estar se perguntando por que o efeito só foi obtido no terço superior da tela. A explicação está no fato de que 255 é o maior valor que um endereço pode conter. Assim, o computador procurará o padrão e a cor de um determinado endereço da tabela de nomes da seguinte forma: o terço superior da tela tem suas cores e padrões nos endereços de 0 a 2055 das tabelas de cores e padrões; o terço médio, dos endereços de 2048 a 4096; e o terço inferior, de 4097 a 6143. Mude as seguintes linhas no programa anterior, para ver o mesmo efeito no terço médio da tela:

```
60 VPOKE BASE(11)+I+2048,RND(1)
*256
70 VPOKE BASE(12)+I+2048,RND(1)
*256
```

No modo gráfico de baixa resolução, a tabela de nomes tem endereço inicial guardado em **BASE (15)**. O número colocado em uma das 768 posições desta tabela é multiplicado por 8, para que obtenhamos a posição da tabela de cores em que estão armazenadas as cores da posição. A tabela de cores tem endereço inicial em **BASE (17)**. Não há tabela de padrões. Mude as seguintes linhas no programa anterior, para compreender melhor o processo:

```
10 SCREEN 3
30 VPOKE BASE(15)+I,0
60 VPOKE BASE(17)+I,RND(1)*256
```

Nas telas de texto, boa parte da memória é utilizada para armazenar os padrões das letras — de uma forma semelhante aos padrões gráficos. Assim, quando colocamos na tabela de nomes um código ASCII, o computador multiplica este valor por 8, para obter a posição em que a forma de letra está armazenada na tabela de padrões. O endereço inicial desta tabela está em **BASE (2)**, na tela de 40 colunas. O programa a seguir mostra na tela estes padrões, depois de lê-los com **VPEEK**.

```
10 SCREEN 0
20 FOR I=65 TO 191
30 VPOKE BASE(0)+1,I
40 FOR J=0 TO 7
50 A=VPEEK(BASE(2)+8*I+J)
55 Z$="00000000"+BIN$(A)
60 LOCATE 10,10+J:PRINT RIGHT$(
Z$,8)
70 NEXT J:FOR K=1 TO 500:NEXT K
80 NEXT I
```

Mude algumas linhas para obter o mesmo resultado na tela de 32 colunas, onde a tabela de padrões fica em **BASE (5)**.

```
10 SCREEN 1
30 VPOKE BASÈ(5)+1,I
50 A=VPEEK(BASE(7)+8*I+J)
```

Na tela de 32 colunas podemos obter também caracteres coloridos. Para isso, existe uma tabela de cores com apenas 32 posições. O número reduzido de posições significa que cada conjunto de oito caracteres terá necessariamente a mesma cor, conforme seu código ASCII — não conforme sua posição na tela.

O programa seguinte demonstra isto, imprimindo todo o conjunto de caracteres e depois enchendo a tabela de cores — **BASE (6)** — com cores aleatórias.

```
10 SCREEN 1
20 FOR I=0 TO 255
30 VPOKE BASE(5)+I,I
40 NEXT I
50 FOR I=0 TO 31
60 VPOKE BASE(6)+I,RND(1)*256
70 NEXT
80 GOTO 50
```

NÃO DESANIME

A organização da memória de vídeo do MSX é complicada. Nossa intenção foi motivá-lo a conhecer melhor seu computador; em outra oportunidade, você poderá compreender com mais clareza a organização desta memória.

MAIS CÓDIGOS DE CONTROLE

Os códigos de controle podem ser usados em um programa para acionar funções especiais e substituir comandos em BASIC. Veja seu emprego em micros de diferentes linhas.

Anteriormente, vimos como utilizar códigos de controle dentro de um programa em BASIC, por meio da função **CHR$**. Esta retorna o caractere correspondente a um determinado código numérico ASCII. Diversos exemplos foram dados para micros da linha TRS-80.

Agora vamos examinar como micros de outras linhas podem trabalhar com códigos de controle.

Alguns computadores não permitem o acesso a todos os códigos de controle. Os da linha Apple II, por exemplo, ao contrário dos da linha TRS-80, têm poucos usos para a função **CHR$** com esta finalidade.

Nos micros da linha Apple, como veremos adiante, a entrada dos códigos de controle é feita comumente por meio do próprio teclado, pois este dispõe de uma tecla <CONTROL>, que é pressionada em combinação com outras.

COMANDOS DO DOS

Emprega-se muito a função **CHR$** para acionar comandos do DOS (sistema operacional de disquetes), a partir de programas em BASIC. O código de controle que possibilita o uso de um comando do DOS em um programa é o 4.

Um comando do DOS em geral só pode ser utilizado de forma direta (sem ser precedido de uma linha de programa). Se quisermos, por exemplo, examinar a lista de arquivos em um disquete, digitamos o comando direto:

```
CATALOG
```

Se usarmos dentro de um programa:

```
100 CATALOG
```

... o comando não funcionará quando o programa for executado. Para que isto ocorra, devemos colocar:

```
100 PRINT CHR$(4)+"CATALOG"
```

Quando esta linha é executada, o interpretador BASIC examina o primeiro caractere da mensagem a ser transmitida para o vídeo, com o comando **PRINT**. Se ele tiver o código ASCII igual a 4, a mensagem será enviada ao sistema operacional, como se fosse uma frase de comando.

ECONOMIZE MEMÓRIA

Existem outras maneiras de se colocar o código 4 em uma linha **PRINT**. O seguinte expediente economiza memória, quando muitas linhas **PRINT** com **CHR$ (4)** são necessárias ao programa:

```
90 LET C4$=CHR$(4)
100 PRINT C4$;"CATALOG"
110 PRINT C4$;"RUN PROG2"
```

Pode-se também pressionar as teclas <CONTROL> e D simultaneamente, logo depois das primeiras aspas que se seguem ao comando **PRINT**, e, em seguida, continuar a digitar normalmente o comando do DOS que se deseja.

Este método apresenta um inconveniente: o código de controle 4 (que é inserido por <**CRTL**> <**D**>) fica "invisível" na listagem, podendo não ser notado posteriormente e nem aparecer na listagem da impressora.

Em geral, o uso da tecla <CONTROL> nos micros Apple dá acesso a uma série de funções via teclado.

Existe uma boa razão para se usar códigos de controle nos micros da linha Spectrum (como o TK-90X): economizar memória. Os códigos podem ser empregados para substituir as instruções **PAPER, INK, BRIGHT** e **FLASH.** São digitados ao se pressionar as teclas <CAPS SHIFT> e <SYMBOL SHIFT> e, em seguida, um número (o que equivale à tecla <CONTROL> de outros micros). O número determinará qual instrução está sendo substituída.

Para **PAPER**, entre simplesmente o número da cor (1 a 7). Por exemplo, usamos o 2 para a cor vermelha.

Para **INK**, pressione <CAPS SHIFT> com o número da cor.

Para obter **BRIGHT**, pressione o número 9 e, para desligá-lo, pressione 8.

Finalmente, para obter **FLASH**, pressione <CAPS SHIFT>, seguido do número 9. Para desligá-lo, use o número 8. Lembre-se de pressionar <CAPS SHIFT> e <SYMBOL SHIFT> primeiro, de cada vez. Veja quanta memória foi economizada em uma linha como esta:

```
10 PRINT PAPER 2;INK 6;"MENU'
```

As palavras-chave **PAPER** e **INK** e os dois pontos e vírgulas ocupam 1 byte de memória cada, enquanto os números 2 e 6 ocupam seis. Usando os códigos de controle, a linha ocupará apenas 1 byte para o par de teclas SHIFT, mais 1 byte para o código de controle. Isto significa uma economia de seis bytes por instrução. Assim, um programa que usasse vários comandos de cor ficaria bem menor em tamanho.

Estes comandos precisam ser digitados dentro de aspas para que funcionem corretamente. Portanto, no exemplo anterior, digite primeiro:

```
10 PRINT "
```

... e em seguida digite os códigos de controle para **PAPER 2** e **INK 6,** conforme foi explicado acima, seguidos pelo restante da linha. Não se esqueça das aspas finais.

Os códigos não ocupam espaço visível na listagem, mas funcionam imediatamente, produzindo texto colorido não só na listagem como na tela.

Pode-se utilizar este recurso simplesmente para enfatizar determinados trechos de uma listagem de programa na tela. Neste caso, coloque os códigos de controle fora das aspas, a não ser que você queira que as cores também apareçam na tela.

INCONVENIENTE

O procedimento descrito acima apresenta um inconveniente: os códigos de controle ficam "invisíveis" na tela e na listagem da impressora, não podendo ser notados posteriormente. Se você quiser mudar alguma cor de frente (**INK**) ou de fundo (**PAPER**) no mesmo programa, precisará apagar a parte inicial da linha (ou até ela toda, se não souber localizar onde estão os códigos de controle) e inserir novos códigos pelo método acima.

COMO MOVIMENTAR O AVENTUREIRO

Para descobrir os mais estranhos lugares sem sair de casa, basta entrar no mundo da aventura. Veja aqui como proceder para dar movimento ao aventureiro e iniciar as explorações.

Você já digitou o programa contendo as descrições dos locais que fazem parte desta aventura? Então deve estar ansioso para fazer com que o jogador comece a explorá-los, possibilitando-lhe deslocar-se de um local para outro. Para que isso aconteça, você precisará definir os movimentos possíveis a partir de um determinado local; deverá estabelecer critérios que permitam julgar as respostas dadas pelo jogador e, finalmente, determinar os caminhos que podem ser percorridos.

Apresentaremos, adiante, algumas rotinas necessárias ao desenvolvimento do programa de aventuras. Estas rotinas serão responsáveis pela correta descrição da localização do jogador e pela apresentação das saídas que lhe serão permitidas. Você digitará cada opção e aprenderá a escrever a seção de programa que movimenta o jogador no mundo da aventura, de acordo com a opção que escolheu.

INÍCIO

É fundamental conhecer a localização exata do jogador, a qualquer momento, dentro do jogo. Para isso, o programa usa uma variável **L**, que indica Local. O valor dessa variável muda de acordo com a localização do jogador, a cada movimento realizado.

Para começar, você deve informar ao computador qual a posição do jogador no início da aventura. A primeira seção de um programa para fazer isso é dada em seguida. Carregue a parte do programa que você já possui, por meio do comando **LOAD**, e adicione estas novas linhas:

```
100 CLS : LET DA=0: LET TA=0:
LET LA=0
270 REM **POSICAO INICIAL**
280 LET L=15
290 GOTO 330
```

```
270 REM **POSICAO INICIAL**
280 L=15
290 GOTO 330
```

O número 15 indica a localização da porta de entrada da Cidade Oculta. Se você desejar que o jogador inicie a aventura em outro local, bastará mudar o valor da variável **L**. Adiante, você verá como ajustar o valor de **L** durante o jogo, de modo que passe a indicar localização diferente. Por enquanto, devemos preparar o programa para receber do jogador informações indicativas do caminho que pretende seguir.

RESPOSTAS

Para que o computador possa compreender e agir corretamente de acordo com as respostas dadas pelo jogador, você deve dar à máquina um conjunto de palavras que ela passe a identificar.

Neste estágio, o computador precisa conhecer apenas as quatro direções, o que pode ser feito por meio do cordão **R$**. Esse cordão é carregado com as palavras indicativas de direção, colocadas em uma instrução **DATA.**

- APRESENTAÇÃO E DESCRIÇÃO DOS LOCAIS
- DIREÇÕES POSSÍVEIS
- A MOVIMENTAÇÃO DO AVENTUREIRO

```
110 REM **PREPARACAO DAS MATRI
ZES DE RESPOSTAS**
120 DIM R$(19,40): DIM R(19)
130 FOR K=1 TO 4: READ R$(K),R
(K): NEXT K
150 DATA "NORTE",1,"SUL",1,"LE
STE",1,"OESTE",1
```

```
110 REM**PREPARAR MATRIZES DAS
RESPOSTAS**
120 DIM R$(19),R(19)
130 FOR K=1 TO 4:READ R$(K),R(K
):NEXT
150 DATA NORTE,1,SUL,1,LESTE,1,
OESTE,1
```

Observe que as matrizes foram dimensionadas na linha 120, podendo agora armazenar todas as respostas que o jogo requer. Como precisamos, inicialmente, de quatro direções, apenas os quatro primeiros elementos das matrizes **R$** e **R** serão utilizados. A instrução **FOR...NEXT,** na linha 130, que lê **R$** e **R** por meio do comando **READ,** varia, assim, de um a quatro. As direções e os valores a ela associados estão colocados na instrução **DATA,** na linha 150.

Essas informações, entretanto, ainda não podem ser usadas pelo jogador. Antes, será necessário dotar o programa de uma rotina que indique onde ele está localizado.

COMO ENCONTRAR O LOCAL

Para que o jogador saiba onde se encontra, após efetuar cada movimento, é necessário fornecer-lhe uma descrição de sua posição. Você já digitou esta des-

crição. Ela está nas linhas de comentários, que se iniciam com a palavra **REM.** Agora, só nos falta uma rotina que associe o número contido na variável **L,** que indica a posição, à descrição correspondente. Neste ponto, os usuários do Spectrum devem digitar uma rotina extra:

```
20 DIM G(11,4): POKE 23658,8
30 FOR N=1 TO 4: FOR M=1 TO
11: READ G(M,N): NEXT M: NEXT
N
40 DATA 0,0,0,5020,0,0,5050,
5080,0,5110,0
50 DATA 5140,0,0,5180,5210,
5240,5270,5300,0,0,0
60 DATA 0,5330,0,5360,0,0,0,0
,0,0,0
70 DATA 1010,1150,1240,1310,
1410,1460,1500,1360,1080,1550
,3110
300 REM **ACHAR O LOCAL**
310 CLS
330 IF L<11 THEN  GOSUB G(L,1)
: GOTO 400
340 IF L<21 THEN  GOSUB G(L-10
,2): GOTO 400
350 IF L<26 THEN  GOSUB G(L-20
,3)
```

```
300 REM **ACHAR LOCAL**
310 CLS
330 IF L<11 THEN ON L GOSUB 0,0
,0,5020,0,0,5050,5080,0,5110:GO
TO 400
340 IF L<21 THEN ON L-10 GOSUB
5140,0,0,5180,5210,5240,5270,53
00,0,0:GOTO 400
350 IF L<26 THEN ON L-20 GOSUB
0,5330,0,5360
```

Antes de escrever esse tipo de rotina, verifique se o número correspondente a cada descrição de localização está correto. Começando com a localização 1, faça uma lista dos números de linhas correspondentes a cada descrição. Se não houver descrição para uma determinada localização, escreva o número 0. Nesta aventura, já dispomos de descrição para a localização de número 4, mas não para as de número 1, 2 e 3.

De posse do conjunto contendo os números de linhas correspondentes à descrição de cada localização, você pode começar a escrever a rotina. Em todos os computadores, exceto o Spectrum, esta rotina é constituída de uma seqüência de operações que verifica o valor da variável **L** e usa o comando **ON...GOSUB.** No caso do Spectrum, que não possui este comando, o número das linhas é obtido de uma matriz.

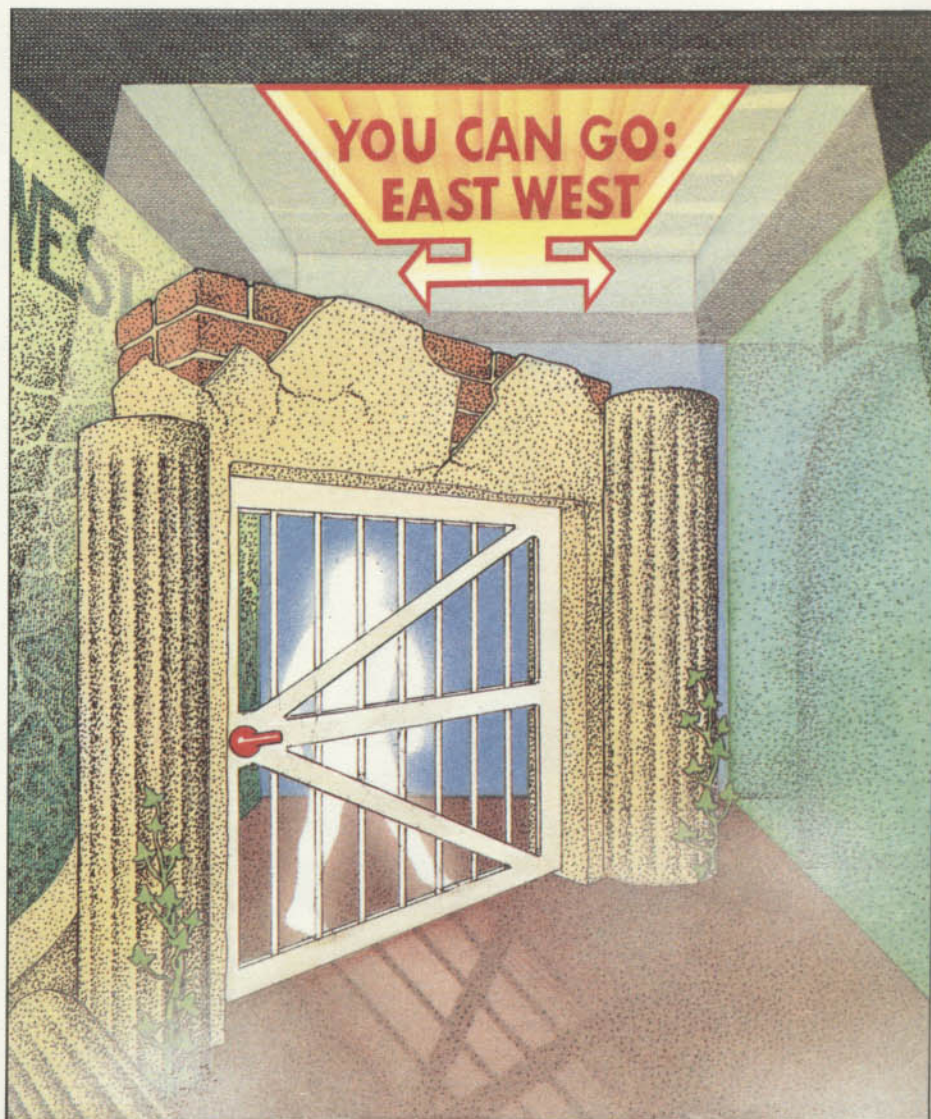
Nos outros programas, o conjunto contendo os números de linhas está nas instruções colocadas nas linhas 330 a 350. Inicia com a localização 1 no começo da linha 330 e termina com a localização 24 no final da linha 350.

No Spectrum, use o valor de **L** para ter acesso a um elemento da matriz construído pelas instruções apresentadas nas linhas 20 e 30. Note que a matriz está superdimensionada em relação à quantidade de locais existentes na aventura. Os números excedentes são todos iguais a zero e não têm nenhuma influência no desenrolar da aventura. O dimensionamento extra foi feito porque a matriz será utilizada em partes do programa que mais tarde apresentaremos.

Uma observação válida somente para os usuários do Spectrum: o comando **POKE,** na linha 20, faz com que o computador aceite apenas letras maiúsculas. Isto evitará problemas no momento de se comparar as respostas dadas pelo jogador com as armazenadas na variável **R$.**

APRESENTAÇÃO DAS DIREÇÕES

Além da descrição do local onde se encontra, o jogador também precisa conhecer as saídas disponíveis. O programa deve verificar, pelas variáveis **N, E, S,** e **W,** quais são estas saídas. A informação é necessária porque nem sempre se pode sair em todas as direções, estando em um determinado local. A próxima seção do programa indica as saídas.

```
390 REM **APONTAR DIRECOES**
```

```
400 IF DA<>1 THEN  PRINT '"POD
E SEGUIR";
410 IF N>0 THEN  PRINT TAB 11;
"NORTE"
420 IF E>0 THEN  PRINT TAB 11;
"LESTE"
430 IF S>0 THEN  PRINT TAB 11;
"SUL"
440 IF W>0 THEN  PRINT TAB 11;
"OESTE"
```

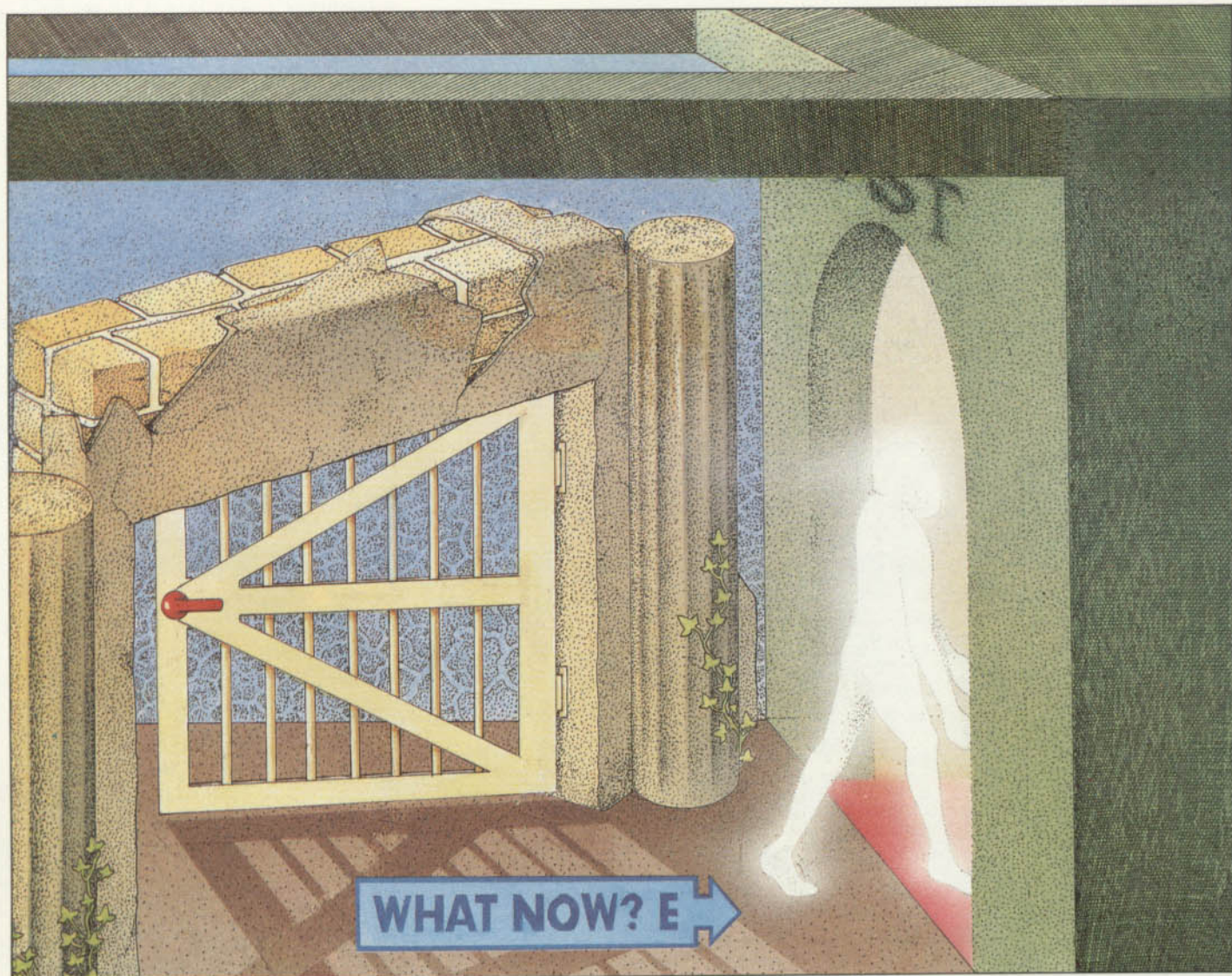
```
390 REM **INDICAR DIRECOES**
400 IF L<>11 OR (LA=1 AND OB(6)
=-1) THEN PRINT:PRINT"PODE SEGU
IR ";ELSE 460
410 IF N>0 THEN PRINT TAB(13);"
NORTE"
420 IF E>0 THEN PRINT TAB(13);"
LESTE"
430 IF S>0 THEN PRINT TAB(13);"
SUL"
440 IF W>0 THEN PRINT TAB(13);"
OESTE"
```

```
390  REM  **APONTAR DIRECOES**
400  IF L <  > 11 OR (LA = 1 AN
D OB(6) =  - 1 THEN  PRINT : PR
INT "PODE SEGUIR ";: GOTO 410
405  GOTO 460
410  IF N > 0 THEN  PRINT  TAB(
 11);"NORTE"
420  IF E > 0 THEN  PRINT  TAB(
 11);"LESTE"
430  IF S > 0 THEN  PRINT  TAB(
 11);"SUL"
440  IF W > 0 THEN  PRINT  TAB(
 11);"OESTE"
```

A rotina apresentada simplesmente verifica os valores das variáveis **N, S, E,** e **W** que você definiu, baseado em seu mapa de localidades. Se o valor da variável for maior do que zero, a direção escolhida é uma direção possível — ou seja, é uma saída. Essa rotina pode ser utilizada, sem modificações, em qualquer aventura que empregue um setoramento desse tipo.

INSTRUÇÕES

Agora que o jogador conhece as direções possíveis, podemos perguntar-lhe: **"PARA ONDE?"**. A rotina responsável por esta pergunta é apresentada abaixo.

```
450 REM **INSTRUCOES**
460 INPUT  INVERSE 1;"PARA OND
E "; LINE I$
470 GOSUB 3010
515 GOTO G(I,4)
```

```
450 REM **INSTRUCOES**
460 PRINT:INPUT"PARA ONDE";I$
470 GOSUB 3010
```

A resposta dada pelo jogador é armazenada na variável **I$**. O programa verificará o tipo de resposta e atuará de acordo.

A próxima linha — linha 470 — desvia o fluxo do programa para uma sub-rotina localizada na linha 3010. Esta sub-rotina é responsável pela validação da resposta do jogador.

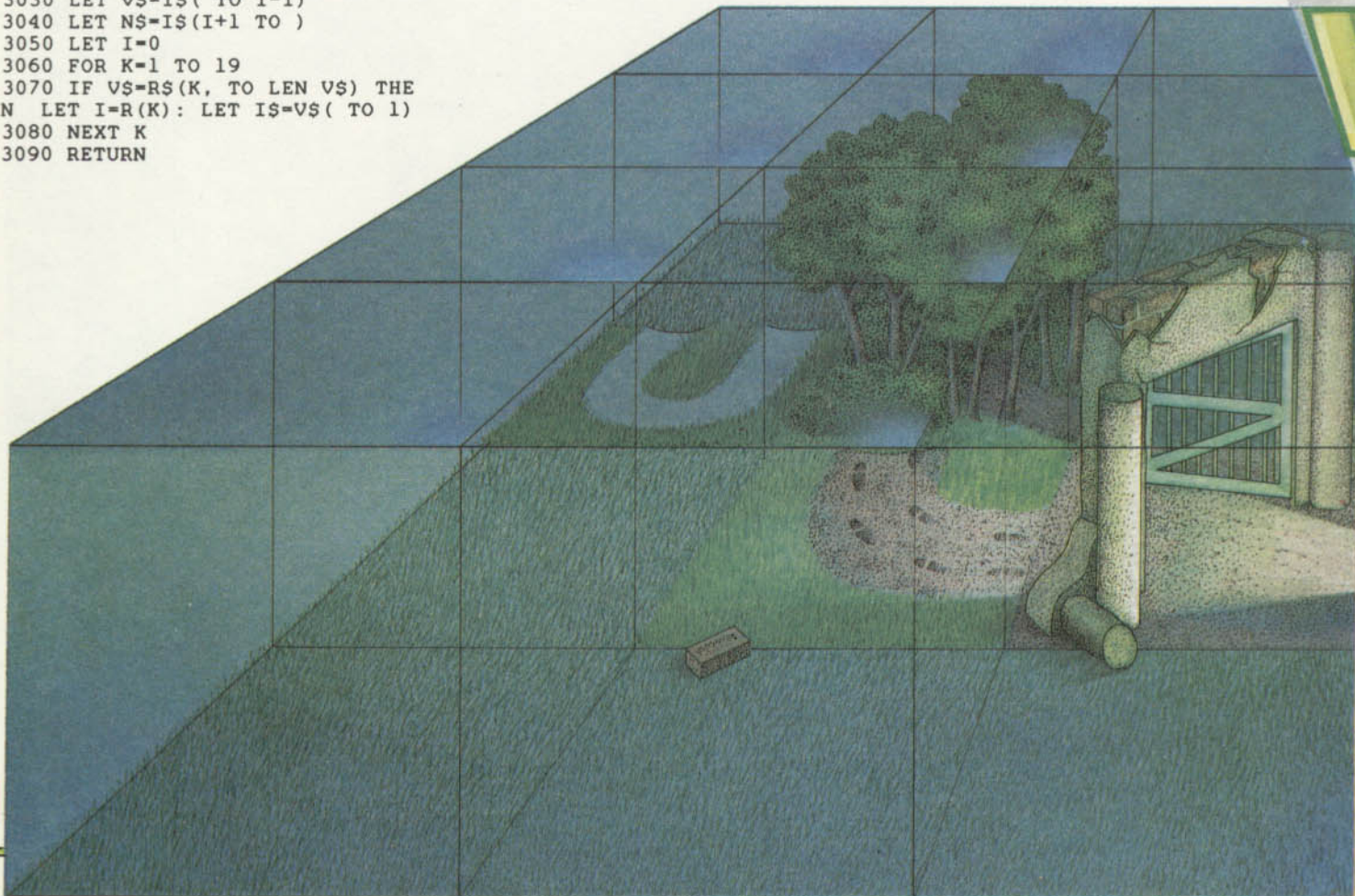
```
 600 REM **ROTINA INSTR.**
 610 LET IN=0: IF LEN Y$>LEN X$
 THEN  RETURN
 620 FOR K=1 TO (LEN X$-LEN Y$+
1)
 630 IF Y$=X$(K TO K+LEN Y$-1)
 THEN  LET IN=K: GOTO 650
 640 NEXT K
 650 RETURN
3000 REM **ROTINA VERIFICACAO**
3010 LET N$="": LET X$=I$: LET
Y$=" ": GOSUB 600: LET I=IN
3020 IF I=0 THEN  LET V$=I$: GO
TO 3050
3030 LET V$=I$( TO I-1)
3040 LET N$=I$(I+1 TO )
3050 LET I=0
3060 FOR K=1 TO 19
3070 IF V$=R$(K, TO LEN V$) THE
N  LET I=R(K): LET I$=V$( TO 1)
3080 NEXT K
3090 RETURN
```

```
3000 REM **INSTRUCAO DE CHECAGE
M**
3010 N$="":I=INSTR(I$," ")
3020 IF I=0 THEN V$=I$:GOTO 305
0
3030 V$=LEFT$(I$,I-1)
3040 N$=MID$(I$,I+1)
3050 I=0
3060 FOR K=1 TO 19
3070 IF INSTR(R$(K),V$)=1 THEN
I=R(K):I$=LEFT$(V$,1)
3080 NEXT
3090 RETURN
```

```
3000  REM  **ROTINA DE VERIFICA
CAO**
3010 N$ = "": FOR Z = 1 TO  LEN
 (I$): IF  MID$ (I$,Z,1) = " "
THEN I = Z: GOTO 3020
3015  NEXT :I = 0
3020  IF I = 0 THEN V$ = I$: GO
TO 3050
3030 V$ =  LEFT$ (I$,I - 1)
3040 N$ =  MID$ (I$,I + 1)
3050 I = 0
3060  FOR K = 1 TO 19
3070  IF V$ =  LEFT$ (R$(K), LE
N (V$)) THEN I = R(K):I$ =  LEF
T$ (V$,1)
3080  NEXT
3090  RETURN
```

A rotina verifica se a variável **I$** compõe-se de duas palavras. Em caso afirmativo, a primeira palavra é chamada de **V$** e a segunda de **N$**. A variável **V$** contém um verbo — como **PEGAR, MATAR, LEVAR** — e todas as palavras indicativas de direção: **NORTE, SUL, LESTE** e **OESTE**. A variável **N$** armazena os nomes dos objetos que fazem parte da aventura.

Os computadores TRS-Color e MSX usam o comando **INSTR**, na linha 3010, para verificar se há algum espaço ha resposta que foi armazenada em **I$**. Este espaço será indicativo da existência de duas palavras: a que pertence a **V$** e a que pertence a **N$**. No Spectrum e no Apple não existe o comando **INSTR**. Para substituí-lo, usamos uma pequena sub-rotina, colocada nas linhas 600 a 650.

Se um espaço for encontrado, a instrução na linha 3030 separa **I$** em suas componentes **V$** e **N$**. Se não houver espaço, a linha 3020 considera **V$** igual a **I$**.

A parte final da sub-rotina, composta pelas linhas 3060 a 3080, compara as respostas dadas com o conteúdo da matriz **R$**. Como sabemos, a matriz **R$** contém as palavras indicativas das direções que podem ser seguidas. Depois você verá como fazer para expandir o conteúdo da matriz **R$**. Pela instrução da linha 3070, o programa verifica se há igualdade entre os conteúdos **R$** e **V$**. Se houver, a variável **I** assume o valor da variável **R**. O programa reconhecerá a igualdade dos conteúdos verificando se o valor de **I** é maior que zero. A última parte da linha 3070 retira a primeira letra de **V$** e armazena-a na variável **I$**. A variável **I$** será utilizada depois, para fazer com que o jogador se movimente.

As sub-rotinas apresentadas adaptam-se a qualquer aventura, sem necessidade de grandes modificações. Apenas um detalhe pode precisar de alguns ajustes: a duração do comando **FOR...NEXT**, na linha 3060.

MOVIMENTOS

O passo seguinte consiste em adicionar uma rotina destinada a manipular a variável **L**, indicativa de localização, de acordo com o valor assumido pela variável **I$**. Esta rotina é apresentada em seguida:

```
1000 REM **ROTINA MOVIMENTO**
1010 IF I$="N" AND N>0 THEN  LE
T L=L-6: GOTO 310
1020 IF I$="L" AND E>0 THEN  LE
T L=L+1: GOTO 310
1030 IF I$="S" AND S>0 THEN  LE
T L=L+6: GOTO 310
1040 IF I$="O" AND W>0 THEN  LE
T L=L-1: GOTO 310
1050 REM **SE NAO HA LOCAL POSS
IVEL NESSA DIRECAO**
1060 PRINT '"DESCULPE, VOCE NAO
 PODE SEGUIR  NESTA DIRECAO.":
GOTO 330
```

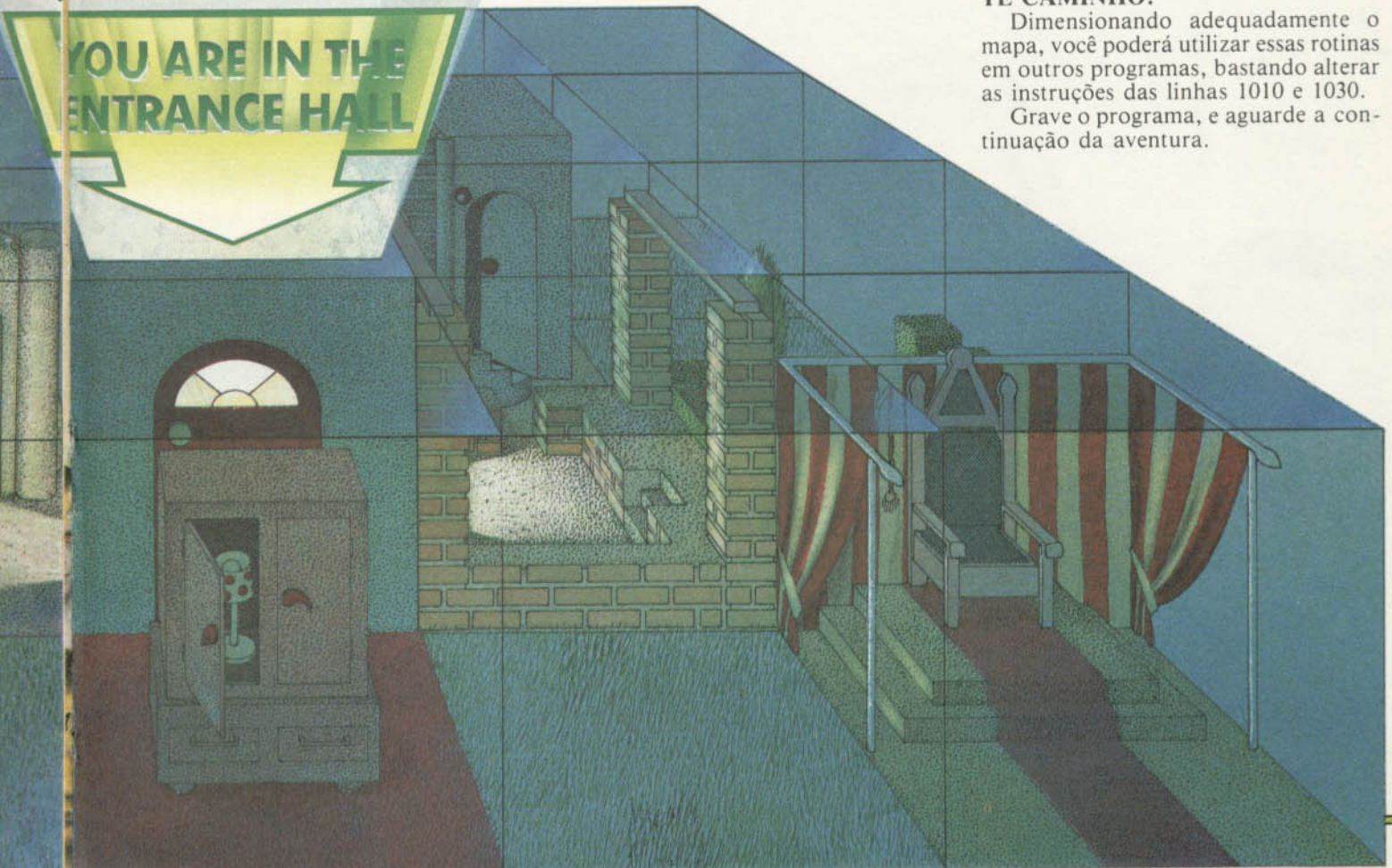
```
1000 REM **ROTINA DE MOVIMENTO*
*
1010 IF I$="N" AND N>0 THEN L=L
-6:GOTO 310
1020 IF I$="L" AND E>0 THEN L=L
+1:GOTO 310
1030 IF I$="S" AND S>0 THEN L=L
+6:GOTO 310
1040 IF I$="O" AND W>0 THEN L=L
-1:GOTO 310
1050 REM **SE NAO HOUVER LOCAL
POSSIVEL NESSA DIRECAO**
1060 PRINT:PRINT"DESCULPE - VOC
E NAO PODE SEGUIR POR ESTE CAMI
NHO.":GOTO 330
```

Como você está lembrado, o ponto de partida da aventura foi um mapa com uma largura de seis posições. Mover o jogador por essas posições significa alterar o valor da variável **L** por um fator baseado no tamanho do mapa. Por exemplo, para fazer com que o jogador se movimente nas direções Norte ou Sul basta adicionar ou subtrair seis da variável **L**. Isso fará com que o jogador suba ou desça uma linha completa no mapa. De modo semelhante, mover o jogador nas direções Leste ou Oeste significa adicionar ou subtrair 1 de **L**.

As instruções das linhas 1010 a 1040 verificam o conteúdo da variável **I$** e ajustam o valor de **L**. As saídas possíveis são definidas nas linhas que seguem as descrições dos locais.

Se não há uma saída na direção que o jogador escolheu, a linha 1060 apresenta a mensagem: **"DESCULPE — VOCÊ NÃO PODE SEGUIR POR ESTE CAMINHO."**

Dimensionando adequadamente o mapa, você poderá utilizar essas rotinas em outros programas, bastando alterar as instruções das linhas 1010 e 1030.

Grave o programa, e aguarde a continuação da aventura.

DATILOGRAFIA: ALFABETO COMPLETO

Se você já dominou o uso das teclas centrais, é hora de treinar a datilografia com todo o alfabeto. Para isso, veja como alterar o programa apresentado no artigo anterior.

No artigo da página 249 apresentamos a primeira parte de um programa completo para aprender datilografia. Tendo já dominado razoavelmente o uso das teclas da fileira do meio — ou seja, se você datilografa cerca de 15 palavras por minuto, sem cometer nenhum erro —, é hora de passar para o segundo estágio.

AS TECLAS QWERTY

Acrescentando as linhas que se seguem ao programa do artigo anterior, você poderá treinar também as teclas da fileira de cima do teclado: a chamada fileira **QWERTY** (é desta fileira que vem o nome dado aos teclados usados em computadores).

Carregue o programa antigo e digite as linhas que se seguem conforme o tipo de seu microcomputador. Algumas delas substituirão linhas previamente existentes — que agora se tornaram desnecessárias —, ao passo que outras são totalmente novas.

S

```
 30 LET S$="QAWSEDRFTGYHUJIKOL
P"
210 FOR K=6 TO 24
230 LET R$=S$(K-5)
320 LET RN=INT (RND*19)+1
330 PRINT AT 10,RN+5,;"*": LET
R$=S$(RN)
350 PRINT AT 10,RN+5;" "
440 LET RN=INT (RND*19)+1
530 PRINT AT 10,13;"
": PRINT AT 10,13;T$
540 FOR M=1 TO LEN T$: PRINT
AT 9,11+M;" *              "
610 FOR N=1 TO 4: RESTORE :
LET RN=INT (RND*24)+1: FOR K=1
TO RN: READ X$: NEXT K
1010 PRINT AT 12,6;S$
2000 DATA "QUEDA","TIRO","TROLE
","POLE","GRALHA","RISO","PORTA
","PATRULHA","URSO","PILHA","RU
A","ILHA"
2010 DATA "TULIPA","PIOLHO","IO.
```

■ COMO ACRESCENTAR AS TECLAS DAS FILEIRAS SUPERIOR E INFERIOR AO PROGRAMA
■ POSIÇÃO DOS DEDOS NAS TECLAS DE OUTRAS FILEIRAS

■ COMO MUDAR AS PALAVRAS PARA TREINAMENTO
■ PRATIQUE COM TODO O ALFABETO

```
DO","PIADA","JAULA","FLORESTA",
"OLHO","PODER","RATO","TROPA","
PISTOLA","QUADRA"
```

```
10 OB$="QAWSEDRFTGYHUJIKOLP"
210 AP=1253
220 FOR K=1 TO 19
230 AP=AP+1
320 AP=1253+RND(19)
430 P$=MID$(OB$,RND(19),1)
800 CLS:P$="":FOR K=1 TO 4
```

```
9000 DATA QUARTO,LADO,FATOS,SER
IA,GALHO,JATO,HARPA,LOTERIA,POR
TA,QUILHA,SADIO,RALA,ORLA,ATILA
9010 DATA EDITAR,TALHER,DIRETO,
ULISSES,ILHA,FOGO,JULGAR,ASSADO
,DERROTA,HULHA,GASODUTO,LATA,TI
JOLO
```

```
10 OB$="QAWSEDRFTGYHUJIKOLPÇ":Z
$=CHR$(219):S$="L10 O2 G"
210 AP=358
220 FOR K=1 TO 20
230 AP=AP+1
320 AP=359+INT(RND(1)*20)
430 P$=MID$(OB$,INT(RND(1)*20)+
1,1)
810 FOR K=1 TO 4:P$=P$+W$(INT(R
ND(1)*28)+1)+CHR$(32):NEXT
9000 DATA QUARTO,LADO,FATOS,SER
IA,GALHO,JATO,HARPA,LOTERIA,POR
TA,QUILHA,SADIO,RAÇA,ORLA,ATIÇA
R
9010 DATA EDITAR,TALHER,DIRETO,
ULISSES,ILHA,FOGO,JULGAR,JULHO,
ASSADO,DERROTA,HULHA,GASODUTO,L
AÇO,TIJOLO
```

```
10 OB$ = "Q A W S E D R F T G Y
 H U J I K O L P ;"
210 AP =  - 1
220  FOR K = 1 TO 20:AP = AP +
2: GOSUB 1100: NEXT
320 AP = 1 +  INT ( RND (1) * 2
0) * 2
420  FOR K = 1 TO 20:P$ =  MID$
 (OB$, INT ( RND (1) * 20) * 2
+ 1,1)
610  FOR K = 1 TO 5:P$ = P$ + W
$( INT ( RND (1) * 28) + 1) +
CHR$ (32): NEXT
1010  VTAB 12: HTAB 1: PRINT OB
$
9000  DATA  QUARTO,LADO,FATOS,
SERIA,GALHO,JATO,HARPA,LOTERIA,
PORTA,QUILHA,SADIO,RALA,ORLA,AT
ILA
9010  DATA  EDITAR,TALHER,DIRE
TO,ULISSES,ILHA,FOGO,JULGAR,JUL
HO,ASSADO,DERROTA,HULHA,GASODUT
O,LACO,TIJOLO
```

Quando o programa for executado, a tela exibirá o menu usual, com cinco opções. O nível 1 exibe a seqüência de letras **QAWSEDRFTGYHUJIKOLP**. Dependendo de seu computador, o teclado também incluirá um caractere final: ; ou **Ç**. Da mesma forma que o programa anterior, um asterisco imediata-

mente abaixo das letras na tela funcionará como um sinal de prontidão, caminhando da esquerda para a direita.

Os níveis 2 e 3 funcionam exatamente como o primeiro, levando você a digitar os caracteres aleatoriamente, mas com uma série muito maior de letras.

Os níveis 4 e 5 são mais difíceis do que os anteriores, porque envolvem a digitação de palavras mais extensas, formadas por letras das duas diferentes fileiras.

Para começar a usar o programa, sente-se diante do teclado e posicione os dedos corretamente na fileira do meio. Tente digitar as teclas da linha superior movimentando apenas o dedo que for necessário para cada uma e não toda a mão (veja a ilustração). Utilize o dedo mínimo da mão esquerda para acionar o **Q** ou o **A**, o dedo anular para o **S** e o **W**, e assim por diante, até o dedo mínimo da mão direita, que deve ser usado para acionar a tecla **P**. Os dedos indicadores trabalharão mais que todos — o indicador esquerdo será utilizado para o **F**, **G**, **R** e **T**, enquanto o indicador direito se encarregará das letras **H**, **J**, **Y** e **U**. Depois de ter pressionado uma tecla da fileira superior, volte com o dedo à posição inicial, na tecla de apoio.

AS TECLAS ZXCV

Quando estiver digitando com precisão e rapidez as teclas da fileira superior juntamente com as da fileira central, comece a usar as teclas da fileira inferior (Z, X, C, V, B, etc.)

Nesta parte do curso, você ainda não trabalhará com as três fileiras. Antes disso, aprenderá a utilizar as teclas das fileiras central e inferior do teclado, simultaneamente, com exercícios em cinco níveis de dificuldade.

Eis aqui as alterações que deverá efetuar:

```
 30 LET S$="AZSXDCFVGBHNJMKL"
210 FOR K=6 TO 21
320 LET RN=INT (RND*16)+1
440 LET RN=INT (RND*16)+1
```

```
 610 FOR N=1 TO 5: RESTORE :
 LET RN=INT (RND*24)+1: FOR K=1
 TO RN: READ X$: NEXT K
1010 PRINT AT 12,6;"S$"
2000 DATA "CASA","FACA","VAZA",
"LAVA","MACA","VACA","BABACA","
JACA","JAZZ","BANDA","BALA","CA
LCA"
2010 DATA "SALVA","CALA","AMA",
"FALA","DA","AFAGA","ALCANCA","
MANHA","CANA","LAZANHA","SAGA",
"CHAMA"
```

```
10 OB$="AZSXDCFVGBHNJMKL"
220 FOR K=1 TO 16
320 AP=1253+RND(16)
430 P$=MID$(OB$,RND(16),1)
800 CLS:P$="":FOR K=1 TO 5
9000 DATA FACA,LAVA,LAMA,BALANC
O,VAGA,NASA,JACA,SAMBA,MACA,AVA
NCA,VANDA,AJAX,CHAMADA,VALHALA,
CALHA,CANSADA
9010 DATA LAMBADA,BANANA,GAMBA,
BANDA,CANA,ABAFA,CANALHA,MASSA,
ZAGA,MANHA,CASCA,SALVA
```

```
10 OB$="AZSXDCFVGBHNJMK,L.Ç":Z$
=CHR$(219):S$="L10 O2 G"
220 FOR K=1 TO 19
320 AP=359+INT(RND(1)*19)
430 P$=MID$(OB$,INT(RND(1)*19)+
1,1)
9000 DATA FACA,LAVA,LAMAÇAL,BAL
ANÇA,VAGA,NASA,JACA,SAMBA,MACA,
AVANÇA,VANDA,AJAX,CHAMADA,VALHA
LA,CALHA,CANSADA
9010 DATA LAMBADA,BANANA,GAMBA,
BANDA,CANA,ABAFA,CANALHA,MASSA,
ZAGA,MANHA,CAÇA,BAÇA
```

```
10 OB$ = "A Z S X D C F V G B H
 N J M K , L . ;"
220  FOR K = 1 TO 19:AP = AP +
2: GOSUB 1100: NEXT
320 AP = 1 +  INT ( RND (1) * 1
9) * 2
420  FOR K = 1 TO 20:P$ =  MID$
 (OB$, INT ( RND (1) * 19) * 2
+ 1,1)
9000  DATA  FACA,LAVA,LAMA,BALA
NCA,VAGA,NASA,JACA,SAMBA,MACA,A
```

```
VANCA,VANDA,AJAX,CHAMADA,VALHAL
A,CALHA,CANSADA
9010 DATA     LAMBADA,BANANA,BAN
DA,CANA,ABAFA,CANALHA,MASSA,ZAG
A,MANHA,CALA,BALA,GAMBA
```

Com esse programa, você também fará exercícios em cinco níveis de dificuldade, mas com palavras que utilizam as letras das fileiras central e inferior do teclado.

Mais uma vez, sente-se diante do teclado, com os dedos posicionados corretamente. Agora você movimentará seus dedos para baixo, para a fileira inferior, antes de retornar às teclas de apoio. O dedo mínimo da mão esquerda irá pressionar o **A** e o **Z**, o anular o **S** e o **X** e assim por diante. O dedo indicador da mão esquerda será utilizado para as letras **F**, **G**, **V** e **B**, e o indicador da mão direita para **H**, **J**, **N** e **M**. Os dedos restantes deverão operar as teclas de pontuação situadas na fileira inferior, em posições que variam de teclado para teclado. No Spectrum, elas não estão disponíveis sem o <**SYMBOL SHIFT**>, cujo emprego será explicado futuramente.

Observe que essas teclas não estão incluídas nos testes de palavras — você deverá praticar a pontuação com um programa adicional, que apresentaremos em artigo posterior.

O ALFABETO COMPLETO

Depois de dominar o uso conjunto das teclas das fileiras superior e inferior do teclado, você estará apto a ingressar no estágio seguinte do curso. Afinal, você poderá praticar a datilografia utilizando todo o alfabeto. Apenas as teclas de números e de pontuação permanecerão temporariamente excluídas do treino.

Eis aqui as alterações que deverá efetuar no programa:

```
 30 LET S$="QAZWSXEDCRFVTGBYHN
 UJMIKOLP"
 40 FOR K=2 TO 27
230 LET R$=S$(K-1)
320 LET RN=INT (RND*26)+1
330 PRINT AT 10,RN+1;"*": LET
R$=S$(RN)
350 PRINT AT 10,RN+1;" "
440 LET RN=INT (RND*26)+1
530 PRINT AT 10,13;"
": PRINT AT 10,13;T$
540 FOR M=1 TO LEN T$: PRINT
AT 9,11+M;" *                "
610 FOR N=1 TO 5: RESTORE :
LET RN=INT (RND*24)+1: FOR K=1
TO RN: READ X$: NEXT K
1010 PRINT AT 12,2;S$
2000 DATA "QUIETO","LONGE","ASI
LO","MENTE","LOCAL","TRADICAO",
"RESPOSTA","ATRAVES","DRIBLE","
RETORNO","DESPEDIDA","ESCRITA"
2010 DATA "ESCOLA","INFERNO","P
ROFESSOR","CHATO","TELEVISAO","
BURRA","ESPORTE","BOM","COMPUTA
DOR","MELHOR","INPUT","PERFEITO
"
```

```
10 OB$="QAZWSXEDCRFVTGBYHNUJMIK
OLP"
210 AP=1248
220 FOR K=1 TO 26
320 AP=1248+RND(26)
430 P$=MID$(OB$,RND(26),1)
800 CLS:P$="":FOR K=1 TO 4
1020 PRINT @257,OB$
9000 DATA MARIA,MULHER,JEITO,CH
EGAR,QUENTE,COSTUME,LOCAL,BONIT
A,DECOTE,CONVERSA,UTERO,JAPAO,C
ORRIDA,AZUL
9010 DATA BANDEIRA,AVIAO,VAZIO,
ABERTO,HALITO,XAXADO,ALCALINO,U
NIDOS,PRESSA,QUERIDA,GUARDIAO,A
BOBORA,NAVIO,REMORSO
```

```
10 OB$="QAZWSXEDCRFVTGBYHNUJMIK
,OL.PÇ":Z$=CHR$(219):S$="L10 O2
G"
210 AP=353
220 FOR K=1 TO 29
320 AP=353+INT(RND(1)*29)
430 P$=MID$(OB$,INT(RND(1)*29)+
1,1)
1010 LOCATE 0,12:PRINTOB$
9000 DATA MARIA,MULHER,JEITO,CH
EGAR,QUENTE,COSTUME,CANÇÃO,BONI
TA,DECOTE,CONVERSA,ÚTERO,JAPÃO,
CORRIDA,AZUL
9010 DATA BANDEIRA,AVIÃO,VAZIO,
ABERTO,HÁLITO,XAXADO,ALCALINO,U
NIDOS,PRESSA,QUERIDA,GUARDIÃO,A
BÓBORA,NAVIO,REMORSO
```

```
10 OB$ = "QAZWSXEDCRFVTGBYHNUJM
IK,OL.P;"
210 AP = 0
220  FOR K = 1 TO 29:AP = AP +
1: GOSUB 1100: NEXT
320 AP = 1 +  INT ( RND (1) * 2
9)
420  FOR K = 1 TO 20:P$ =  MID$
 (OB$, INT ( RND (1) * 29) + 1,
1)
9000  DATA  MARIA,MULHER,JEITO,
CHEGAR,QUENTE,COSTUME,CANCAO,BO
NITA,DECOTE,CONVERSA,UTERO,JAPA
O,CORRIDA,AZUL
9010  DATA  BANDEIRA,AVIAO,VAZI
O,ABERTO,HALITO,XAXADO,ALCALINO
,UNIDOS,PRESSA,QUERIDA,GUARDIAO
,ABOBORA,NAVIO,REMORSO
```

Como acontece com todos os programas deste curso, assim que você estiver familiarizado com as palavras constantes nas declarações **DATA**, poderá trocá-las por novas palavras. Quase sempre, os programas para outras máquinas incluem palavras diferentes. Se quiser, aproveite-as em seu computador, mas verifique se não está utilizando um número de palavras menor que o do programa original. Caso isso ocorra, você obterá uma mensagem de erro **OUT OF DATA**. E, se colocar mais palavras do que as existentes, elas não serão lidas pelo computador, a menos que você modifique o programa.

Siga as cinco lições, como nos estágios anteriores. Lembre-se sempre de colocar os dedos de volta nas posições corretas de apoio, na fileira do meio, toda vez que pressionar uma tecla das carreiras inferior ou superior.

APERFEIÇOAMENTO

Na medida em que o curso for progredindo, você terá a oportunidade de se aperfeiçoar cada vez mais, adquirindo velocidade e precisão na datilografia com todas as letras do alfabeto. Em seguida, começará a praticar com as teclas de números e de pontuação — essenciais para a digitação das listagens de programas.

Como acentuar textos em português em um microcomputador?

Depende muito da linha ou da marca do micro. O problema não foi tecnicamente resolvido de maneira uniforme pelos fabricantes nacionais, pois durante muito tempo não houve um padrão industrial obrigatório. Entre os computadores pessoais cobertos por INPUT, apenas os micros da linha MSX seguem o padrão atual de representação dos sinais característicos da língua portuguesa — o chamado BRASCII, que é a extensão brasileira do código ASCII.

O BRASCII, além de definir os códigos numéricos para as letras acentuadas (á, à, Á, À, ô, etc.), determina ainda a localização padronizada da cedilha e dos acentos grave, agudo, til e circunflexo, no teclado do microcomputador. Portanto, a maneira de usá-los, na datilografia, é idêntica à de uma máquina de escrever. Da mesma forma, o comportamento do teclado e do vídeo, durante a datilografia das letras acentuadas, deve ser igual: por exemplo, ao se pressionar a tecla com o til, este sinal aparece no vídeo, e o cursor fica parado no mesmo lugar; ao se pressionar a letra a, ela aparece imediatamente abaixo do acento.

Os micros de outras linhas (Apple II, TRS-80, TRS-Color e Sinclair), por serem copiados de modelos norte-americanos e ingleses, dão ao problema respostas diferentes. As soluções adotadas variam de um teclado inteiramente compatível com o de uma máquina de escrever (por exemplo, o Microengenho II, da Spectrum, que pertence à linha Apple), até a impossibilidade total de acentuação (micros compatíveis com as linhas Sinclair ZX-81 e TRS-Color).

Alguns micros brasileiros da linha Apple adotaram o padrão do chamado *teclado inteligente*, ou *teclado profissional*, em que certas teclas são usadas para digitar a letra já acentuada, com uma única pressão. A localização destas teclas não coincide, evidentemente, com a de uma máquina de escrever, e seu acionamento depende da pressão simultânea de uma ou mais teclas adicionais de controle. Este procedimento dificulta a datilografia, pois difere muito da maneira natural de se usar uma máquina de escrever e compromete bastante a velocidade da digitação.

MELHORE A SUA DATILOGRAFIA

- ACRESCENTE NÚMEROS
- A TECLA <SHIFT>
- DESENVOLVA UM RITMO REGULAR
- VELOCIDADE E PRECISÃO

Divirta-se com um joguinho de ação rápida e aprenda ao mesmo tempo a trabalhar com as teclas de números, ampliando a sua habilidade como datilógrafo.

Depois de ter estudado as duas primeiras lições de datilografia, você já está provavelmente familiarizado com as teclas de letras e com os sinais de pontuação das três fileiras inferiores do teclado. Até agora, porém, não tocamos nas importantes teclas numéricas (muito usadas, aliás, na digitação de programas.)

Da mesma forma, ainda não ensinamos as formas de manipulação das teclas para se obter letras maiúsculas ou minúsculas, nem explicamos os sinais de pontuação adicionais e demais símbolos disponíveis no teclado.

Esta lição será dedicada às teclas ainda não estudadas. Nela, apresentaremos também um pequeno e divertido exercício, que fará com que você melhore a sua velocidade, precisão e ritmo no trabalho com o teclado.

ACRESCENTANDO OS NÚMEROS

Para treinarmos a datilografia com a fileira do teclado que contém os algarismos, basta fazer algumas modificações no programa apresentado nas lições anteriores do curso (páginas 253 e 276).

Desta forma, carregue o último programa utilizado e digite as linhas adicionais, expostas a seguir. Como você verá, algumas delas irão apenas substituir linhas já existentes, ao passo que outras serão adicionadas ao programa:

```
 30 LET S$="1A2S3D4F5G6H7J8K9L
 0"
210 FOR K=6 TO 24
230 LET R$=S$(K-5)
320 LET RN=INT (RND*19) 1
330 PRINT AT 10,RN+5;"*": LET
R$=S$(RN)
350 PRINT AT 10,RN+5;" "
 440 LET RN=INT (RND*19)+1
 530 PRINT AT 10,13;"
 ": PRINT AT 10,13;T$
 540 FOR M=1 TO LEN T$: PRINT
 AT 9,11+M;" *              "
 610 FOR N=1 TO 5: RESTORE :
 LET RN=INT (RND*24)+1: FOR K=1
 TO RN: READ X$: NEXT K
1010 PRINT AT 12,6;S$
2000 DATA "MC6809E","VALOR","UL
TIMO","Z80A","RELAXE","6502","A
37XZ","1024","VASTO","JUNCO","R
ETORNO","67VDG"
2010 DATA "APENAS","AGITADO","7
4LS83","REDONDO","LINEAR","1986
","30123","CONGELADO","4MHZ","W
X101","64MB","VIDEO"
```

```
10 OB$="1A2S3D4F5G6H7J8K9L0:-;"
210 AP=1252
220 FOR K=1 TO 22
320 AP=1252+RND(22)
430 P$=MID$(OB$,RND(22),1)
1020 PRINT @261,OB$
9000 DATA MC6809E,VALOR,"ULTIMO
:",Z80A,RELAXE,6502,A37XZ,-1024
,VASTO,JUNCO
9010 DATA RETURN,A847VDG,145,22
,"APENAS,",AGITADO,74LS83,-93.4
1,REDONDO,LINEAR
9020 DATA PROCESSO,CONGELADO,TR
AZ,1986,DESENHO,SANGUE,842.52,"
301,123",350KG
```

```
10 OB$ = "1A2S3D4F5G6H7J8K9L0;:
-"
210 AP = 8
220  FOR K = 1 TO 22:AP = AP +
1: GOSUB 1100: NEXT
320 AP = 9 +  INT ( RND (1) * 2
3)
420  FOR K = 1 TO 20:P$ =  MID$
 (OB$, INT ( RND (1) * 22) + 1,
1)
640 P = 1: VTAB 11: HTAB 2: PRI
NT P$
660  VTAB 15: HTAB P + 1: PRINT
  MID$ (P$,P,1)
1010  VTAB 12: HTAB 9: PRINT OB
$
9000  DATA  MC6809E,VALOR,"ULTI
MO:",Z80A,RELAXE,6502,A37XZ,-10
24,VASTO,JUNCO
9010  DATA  RETURN,A847VDG,145.
22,"APENAS,",AGITADO,74LS83,-93
.41,REDONDO,LINEAR
9020  DATA   PROCESSO,CONGELADO
,TRAZ,1986,DESENHO,SANGUE,842.5
2,"301,23",350KG
```

```
10 OB$="1A2S3D4F5G6H7J8K9L0Ç-="
:Z$=CHR$(219):S$="L10 O2 G"
210 AP=355
220 FOR K=1 TO 22
320 AP=356+INT(RND(1)*22)
430 P$=MID$(OB$,INT(RND(1)*22)+
1,1)
1010 LOCATE 2,12:PRINTOB$
9000 DATA MC6809E,VALOR,"ULTIMO
:",Z80A,RELAXE,6502,A37XZ,-1024
,VASTO,JUNCO
9010 DATA RETURN,A847VDG,145.22
,"SOMENTE,",AGITAR,74LS83,-93.4
1,CÍRCULO,LINEAR
9020 DATA PROCESSO,FRIO,TRAZ,19
86,DESENHO,SANGUE,842.52,CAÇA,3
50KG
```

Quando o programa for rodado, um menu contendo os cinco níveis já conhecidos será apresentado na tela; você deverá então optar por um deles. Como das vezes anteriores, existirão muitas combinações possíveis, dependendo do

número de teclas a serem incluídas nos exercícios. Nos níveis mais baixos de dificuldade, o computador proporá exercícios em que as teclas numéricas e, algumas vezes, de pontuação serão combinadas aleatoriamente com as outras teclas, que você já conhece das duas primeiras lições. Essa combinação dificultará as coisas para você, e o obrigará a não se concentrar apenas nas teclas numéricas.

Nos níveis mais altos de dificuldade, o computador pedirá que você digite uma mistura de palavras, grupos de números e combinações de letras e algarismos. As palavras e números selecionados se encontram nas declarações **DATA**, quase no final do programa. Eles podem ser trocados, caso você queira aumentar o desafio depois de se familiarizar bem com os exercícios originais. Entretanto, lembre-se de manter o mesmo número de palavras (ou grupo de caracteres); do contrário, ocorrerá um erro de execução quando o programa não encontrar o número correto de itens em **DATA**.

Uma vez dominadas as teclas numéricas, passe para a lição seguinte, cujo objetivo é praticar com os caracteres acessíveis apenas por intermédio da tecla <**SHIFT**>.

AS LETRAS MAIÚSCULAS

A tecla <**SHIFT**> corresponde à alavanca de maiúsculas de uma máquina de escrever comum. Em alguns computadores, essa função também pode ser desempenhada pela <**CAPS LOCK**>. Para aprender a usá-las, adicione as linhas abaixo à última versão do programa (novamente, algumas linhas serão inteiramente substituídas):

```
 20 POKE 23658,0: LET ER=0
 30 LET S$="A!aS@sD#dF$fG%gH&h
 J'jK(kL)I0"
 210 FOR K=2 TO 29
 230 LET R$=S$(K-1)
 320 LET RN=INT (RND*28)+1
 330 PRINT AT 10,RN+1;"*": LET
 R$=S$(RN)
 350 PRINT AT 10,RN+1;" "
 440 LET RN=INT (RND*28)+1
 610 FOR N=1 TO 4: RESTORE :
 LET RN=INT (RND*24)+1: FOR K=1
 TO RN: READ X$: NEXT K
1010 PRINT AT 12,2;S$
2000 DATA "$235.50","PRINT#","&
H1200","23.5%","Conta","LONDRES
","Eles","15@&12","(abaixo)","H
+9=1D","**Obs**","Fogo!!"
2010 DATA ";-;;-;","Extra","Bei
ja-flor","Lugar","4*4=16","Quem
?","6:10pm","Nos","$15.40","Dir
igir","ATENCAO","100/4"
```

```
10 OB$="!A"+CHR$(34)+"S#D$F%G&H
'J(K)L*=+"
20 POKE 282,0:CLS
210 AP=1250
220 FOR K=1 TO 21
320 AP=1250+RND(21)
430 P$=MID$(OB$,RND(21),1)
999 POKE 282,255:CLS:END
1020 PRINT @259,OB$
9000 DATA PRINT#,Mostre,&H4000,
Fora!!,(abaixo),H+9=1D,$500.10,
D/100%,Eles,**obs**
9010 DATA Extra,Carga,DIARIO,Co
nta,Mes,Resposta,Hoje,Gerente,S
etor
9020 DATA LONDRES,Concha,Toque,
;-;;-;,Sucesso,Beija-flor,Lugar
,Campo,Direto
```

```
10 OB$ = "!A" +  CHR$ (34) + "S
#D$F%G&H'J(K)L+*=<>?"
220  FOR K = 1 TO 24:AP = AP +
1: GOSUB 1100: NEXT
320 AP = 9 +  INT ( RND (1) * 2
5)
```

```
420  FOR K - 1 TO 20:P$ -  MID$
 (OB$, INT ( RND (1) * 24) + 1,
1)
9000  DATA  PR#,Mostre,$H4000,F
ora!!,(abaixo),H + 9 - 1D,Cz$ 5
00.10,D/100%,Eles,**Obs**
9010  DATA  Extra,Carga,Diario,
Conta,Mes,Hoje,Resposta,Gerente
,Setor
9020  DATA  LONDRES,Concha,Toqu
e,;-;;-;,Sucesso,Beija-flor,Lug
ar,Campo,Direto
```

```
10 OB$="!A@S#D$F%G"+CHR$(34)+"H
&J*K(L)Ç_+?>":Z$=CHR$(219):S$="
L10 O2 G"
220 FOR K=1 TO 24
320 AP=356+INT(RND(1)*24)
430 P$=MID$(OB$,INT(RND(1)*24)+
1,1)
9000 DATA PRINT#,Mostra,&H4000,
Fora!!,(abaixo),H + 9 - 1D,Cz$
500.10,D/100%,Eles,** Obs **
9010 DATA Extra,Cargo,DIARIO,Co
nta,Mês,Resposta,Hoje,Gerente,S
eção
9020 DATA LONDRES,Concha,Toque,
;?;?,Sucesso,Beija-flor,Lugar,C
ampo,Guia
```

O TRS-Color não imprime letras minúsculas na tela. Em vez disso, exibe-as em vídeo inverso (letras claras sobre fundo escuro).

Nos micros da linha Apple que dispõem de minúsculas, mude a tecla seletora para minúsculas ao digitar as linhas **DATA**. Infelizmente, os micros da linha Apple II+ padrão, que não contam com letras minúsculas, não conseguirão rodar o programa a seguir. Nos micros da linha MSX, certifique-se de que a tecla <**CAPS**> está desativada.

Os níveis mais baixos de dificuldade do programa apresentam agora os caracteres que só estão disponíveis quando as teclas são pressionadas simultaneamente com o comando <**SHIFT**>: pontuação, símbolos matemáticos, etc... Eles estão misturados com as letras da fileira central do teclado, o que significa que você deve retornar a eles toda vez que pressioná-los.

Nos níveis mais altos são apresentados palavras e grupos de caracteres de uma lista interna, tal como antes; agora, porém, você encontrará as letras maiúsculas e outros símbolos que só podem ser obtidos com o <**SHIFT**>, misturados com as letras minúsculas.

Conhecidos esses exemplos particulares de teste, substitua as declarações **DATA** por um novo conjunto de dados (lembre-se de manter o mesmo número total de palavras).

Passe para a próxima seção apenas quando estiver encontrando — sem hesitação e sem olhar para o teclado — todos os caracteres com que treinou.

JOGO DE VELOCIDADE

Um dos modos mais eficazes de se melhorar a precisão e a velocidade de datilografia consiste em pressionar as teclas ao compasso de um metrônomo (instrumento que serve para regular os andamentos musicais). Este deve ser acelerado à medida que o movimento de seus dedos se tornar mais rápido e preciso.

Mas por que dar-se ao trabalho de utilizar um metrônomo, quando o seu computador possui um relógio embutido? O próximo programa é um novo e completo exercício de digitação, planejado como um jogo, onde a contagem depende do seu desempenho no teclado. Ele é composto por duas partes. A primeira, exibe uma linha de caracteres selecionada aleatoriamente — você tem que digitar, em seqüência, os caracteres à medida que forem aparecendo. A segunda parte é mais difícil — agora, os caracteres são lançados aleatoriamente na tela, um por um; deste modo, você não tem idéia do que virá a seguir.

Antes de iniciar o teste, selecione o seu próprio nível de dificuldade. Isso deve ser feito dizendo-se ao computador com que velocidade você quer que as letras sejam exibidas — em outras palavras, quantos caracteres por minuto você quer digitar. Então, o computador estabelecerá um tempo limitado, dentro do qual você deve digitar cada caractere; do contrário, será emitida uma contagem de erros. No primeiro nível, isso é feito por intermédio de um indicador móvel que mostra qual letra deveria estar sendo digitada; no segundo nível, o caractere é iluminado por algum tempo.

Ao começar o primeiro teste, você pode escolher se quer o teclado normal (com letras, apenas) ou teclado completo (com todos os símbolos). E antes de passar para o segundo nível (o teste dos caracteres) você deve decidir quanto tempo é capaz de sustentá-lo. O computador perguntará quantos caracteres, no

total, você quer que façam parte do teste.

Imediatamente depois de ter exibido todos os caracteres, o computador parará e fornecerá uma contagem baseada nos seus erros.

Não é apenas a velocidade total de digitação no computador que conta nesse teste de desafio: você precisará também desenvolver um ritmo constante. Para ajudá-lo a adquirir esse ritmo, o computador dará um sinal sonoro assim como um sinal visual de prontidão para cada letra.

Agora digite o programa propriamente dito, e teste a sua habilidade como datilógrafo:

```
10 BORDER 7: PAPER 7: INK 0:
CLS
20 LET a$="ABCDEFGHIJKLMNOPQR
STUVWXYZ"
30 LET a$=a$+"abcdefghijklmno
pqrstuvwxyz"
40 LET a$=a$+"1234567890!@#$%
&'()"
50 LET a$=a$+CHR$ (34)+"<>;-+
=^?/*,."
60 PRINT  INVERSE 1;AT 6,7;"
TESTE 1 OU 2 ? "
70 IF INKEY$="" THEN  GOTO 70
80 LET i$=INKEY$: IF i$="2"
THEN  GOTO 400
90 IF i$<>"1" THEN  GOTO 70
100 CLS : INPUT "Quantos carac
teres por minuto? ";cpm
110 LET t=3000/cpm
120 LET s$=""
130 FOR n=1 TO 30
140 LET s$=s$+a$(INT (RND*84)+
1)
150 NEXT n
160 PRINT  BRIGHT 1;AT 11,1;s$
200 GOSUB 800: LET er=0: FOR r
=1 TO 30
210 POKE 23672,0: POKE 23673,0
220 PRINT AT 10,r-1;" *"
230 SOUND .02,20
240 IF  PEEK  23672+256*PEEK
23673>=t THEN  LET er=er+1:
GOTO 300
250 LET i$=INKEY$: IF i$=""
THEN  GOTO 240
260 IF i$=s$(r) THEN  PRINT AT
12,r;"^": GOTO 280
270 LET er=er+1
280 IF  PEEK  23672+256*PEEK
23673<t THEN  GOTO 280
300 NEXT r
310 PRINT AT 16,3;"VOCE ERROU
";er;" DAS 30"
320 FOR f=1 TO 200: NEXT f
330 GOTO 20
400 CLS : INPUT "Numero de tec
las por minuto? ";cpm
410 INPUT "Numero de caractere
s?";r
420 INPUT "(N)ormal ou (E)sten
dido? "; LINE m$
430 IF m$="N" OR m$="n" THEN
LET a$=a$( TO 52): GOTO 450
440 IF m$<>"E" AND m$<>"e"
THEN  GOTO 420
450 LET t=3000/cpm
```

```
460 GOSUB 800
470 LET er=0
480 FOR n=1 TO r
490 POKE 23672,0: POKE 23673,0
500 LET r$=a$(INT (RND*LEN a$)
+1)
510 PRINT  INVERSE 1;AT 10,15;
r$; INVERSE 0;AT 11,15;" "
520 SOUND .02,20
530 IF  PEEK  23672+256*PEEK
23673>=t THEN  LET er=er+1:
GOTO 580
540 LET i$=INKEY$: IF i$=""
THEN  GOTO 530
550 IF i$=r$ THEN  PRINT AT 11
,15;"^": GOTO 570
560 LET er=er+1
570 IF PEEK 23672+256*PEEK
23673<t THEN  GOTO 570
580 NEXT n
590 PRINT AT 16,3;"VOCE ERROU
";er;" DAS ";n-1
600 FOR f=1 TO 200: NEXT f
610 GOTO 20
800 LET c$="5..4..3..2..1..0"
810 FOR n=1 TO 16
820 PRINT AT 2,5+n;c$(n)
830 PAUSE 10
840 NEXT n
850 SOUND .2,10
860 PRINT AT 2,0;TAB 31;" "
870 RETURN
```

```
10 CLS
20 PRINT @70,"QUAL TESTE (1 OU
2)?"
30 PRINT @102,"DIGITE (0) PARA
SAIR"
40 A$=INKEY$:IF A$<"0" OR A$>"2
" THEN 40
50 ON VAL (A$)+1 GOSUB 1000,600
,200
60 POKE 282,255
70 ER=0:W$="":B$=""
80 GOTO 20
200 CLS:INPUT"DIGITE QUANTAS TE
CLAS PRESSIONADAS POR MINUTO";K
P
210 IF KP<1 THEN 200
220 INPUT "DIGITE O NUMERO DE C
ARACTERES ";NC
230 IF NC<1 THEN 220
240 NM=NC
250 PRINT:PRINT"NORMAL OU ESTEN
DIDO (N/E)?"
260 A$=INKEY$:IF A$<>"N" AND A$
<>"E" THEN 260
270 RN=90:ST=32:IF A$="N" THEN
RN=58:ST=64
280 POKE 282,0
290 TM=3000/KP
300 CLS0:PRINT" PRESSIONE A TEC
LA DEPOIS DO BIP "
310 PRINT @238,"     ";:PRINT @27
0,"     ";:PRINT @302,"     ";
320 W$=CHR$(RND(RN)+ST)
330 IF W$>"Z" AND W$<"a" THEN 3
20
340 PRINT @271,W$;
350 TIMER=0
360 A$=INKEY$:IF A$="" THEN 380
370 B$=A$:IF B$=W$ THEN SCREEN
0,1
380 IF TIMER <TM THEN 360
390 SOUND 150,1:IF B$="" THEN 3
50
400 W$=CHR$(RND(RN)+ST)
410 IF W$>"Z" AND W$<"a" THEN 4
00
420 PRINT @271,W$;
430 TIMER=0
440 A$=INKEY$:IF A$="" THEN 460
450 B$=A$:IF B$=W$ THEN SCREEN
0,1
460 IF TIMER<TM THEN 440
470 SOUND 150,1
480 IF B$<>W$ THEN ER=ER+1:GOTO
 490
490 POKE 1295,128
500 NC=NC-1:IF NC>0 THEN 400
510 CLS:PRINT @448,"COM";KP;"TE
CLAS PRESSIONADAS POR MINUTO"
520 PRINT "VOCE ERROU";ER;"DAS"
;NM
530 RETURN
600 CLS:INPUT"DIGITE QUANTAS TE
CLAS PRESSIONADAS POR MINUTO";K
P
610 IF KP<1 THEN 600
620 PRINT:PRINT"NORMAL OU ESTEN
DIDO (N/E)?"
630 A$=INKEY$:IF A$<>"N" AND A$
<>"E" THEN 630
640 RN=91:ST=31:IF A$="N" THEN
RN=58:ST=64
650 TM=3000/KP
660 CLS:POKE 282,0
670 FOR K=1 TO 32
680 CR=RND(RN)+ST
690 IF CR>90 AND CR<97 THEN CR=
32
700 W$=W$+CHR$(CR)
710 NEXT
720 AP=1248
730 POKE AP,106
740 PRINT @256,W$
750 TIMER=0
760 A$=INKEY$:IF A$="" THEN 780
770 B$=A$
780 IF TIMER<TM THEN 760
790 SOUND 150,1:IF B$="" THEN 7
50
800 TIMER=0
810 A$=INKEY$:IF A$="" THEN 830
820 B$=A$
830 IF TIMER<TM THEN 810
840 SOUND 150,1
850 IF B$<>MID$(W$,AP-1247,1) T
HEN ER=ER+1:GOTO 860
855 POKE AP+64,94
860 POKE AP,96
870 AP=AP+1
```

```
880 IF AP=1280 THEN 910
890 POKE AP,106
900 B$="":GOTO 800
910 CLS:PRINT @481,"COM";KP;"TE
CLAS POR MINUTO"
920 PRINT"VOCE ERROU";ER;"DAS 3
2";:RETURN
1000 CLS
```

```
10 CLS:Z$=CHR$(219):R=RND(-TIME
)
20 LOCATE 5,3:PRINT"QUAL TESTE?
 (1-2)"
30 LOCATE 5,4:PRINT"DIGITE <0>
PARA TERMINAR"
40 A$=INKEY$:IFA$<"0"ORA$>"2"TH
EN40
50 ON VAL(A$)+1 GOSUB 1000,600,
200
60 ER=0:W$="":B$=""
70 GOTO20
200 CLS:INPUT"TOQUES POR MINUTO
";KP
210 IFKP<1THEN200
220 INPUT"NÚMERO DE CARACTERES"
;NC
230 IFNC<1THEN220
240 NM=NC
250 PRINT:PRINT"TECLADO: LETRAS
 OU TOTAL (L/T)?"
260 A$=INKEY$:IFA$<>"L"ANDA$<>"
T"THEN260
270 RN=90:ST=33:IFA$="L"THENRN=
58:ST=65
280 TM=3600/KP
300 CLS:PRINT"PRESSIONE A TECLA
 APÓS O BIP"
310 LOCATE15,10:PRINTSTRING$(4,
219)
320 LOCATE15:PRINTZ$;"  ";Z$
330 LOCATE15:PRINTZ$;"  ";Z$
340 LOCATE15:PRINTSTRING$(4,219
)
350 W$=CHR$(RND(1)*RN+ST)
360 IF W$<"Z"ANDW$>"a"THEN350
370 LOCATE17,12:PRINTW$;:TIME=0
380 A$=INKEY$:IFA$=""THEN390ELS
EB$=A$:IFB$=W$THENCOLOR15,6
390 IFTIME<TMTHEN380ELSEBEEP:IF
B$=""THEN370
400 IFB$<>W$THENER=ER+1
410 COLOR 15,4,4
420 NC=NC-1:IFNC>0THEN350
430 CLS:LOCATE2,12:PRINT"A";KP;
"toques por minuto, você errou"
;ER;"em";NM
440 RETURN
600 CLS:INPUT"TOQUES POR MINUTO
";KP
610 IFKP<1THEN600
620 PRINT:PRINT"TECLADO: LETRAS
 OU TOTAL? (L/T)"
630 A$=INKEY$:IFA$<>"L"ANDA$<>"
T"THEN630
640 RN=91:ST=32:IFA$="L"THENRN=
58:ST=65
650 TM=3600/KP
660 CLS:FORK=1TO32
670 CR=INT(RND(1)*RN)+ST
680 IFCR>90ANDCR<97THENCR=32
690 W$=W$+CHR$(CR)
700 NEXT
710 AP=564
720 VPOKEBASE(0)+AP,205
730 LOCATE3,15:PRINTW$
740 TIME=0
750 A$=INKEY$:IFA$=""THEN760ELS
EB$=A$
760 IFTIME<TMTHEN750
770 BEEP:IFB$=""THEN740
780 IFB$<>MID$(W$,AP-563,1)THEN
ER=ER+1ELSEVPOKEBASE(0)+AP+80,1
790 VPOKEBASE(0)+AP,0:AP=AP+1:I
FAP=596THEN810
800 VPOKEBASE(0)+AP,205:GOTO740
810 CLS:LOCATE2,15:PRINT"A";KP;
"toques por minuto, você errou"
;ER;"de 32"
820 RETURN
1000 CLS
```

COMO AUMENTAR A VELOCIDADE DE DIGITAÇÃO

Convém começar a execução do programa de jogo de velocidade com o teclado normal e a uma velocidade baixa (por exemplo, de cerca de trinta a cinqüenta caracteres por minuto). Ao mesmo tempo, procure não alterar o ritmo de trabalho, digitando com a mesma velocidade tanto os caracteres conhecidos como os que lhe são menos familiares (esse conselho vale principalmente para quando você estiver com o teclado completo).

Se o seu ritmo se mantiver constante, você poderá selecionar o teclado completo e aumentar, gradativamente, a velocidade de digitação.

Outra coisa muito importante para quem está aprendendo a datilografar é fazer todos os exercícios propostos *sem olhar* para o teclado do computador.

Mantenha os seus olhos fixos todo o tempo na tela à sua frente, e repouse as mãos sobre o teclado, distribuindo os dedos pelas teclas de apoio da fileira central, conforme ensinamos na primeira lição.

FAÇA TODOS OS EXERCÍCIOS

Para adquirir um conhecimento profundo de todas as teclas de caracteres pratique todos os exercícios apresentados até agora. Esta parte do curso é independente. Entretanto, em um artigo posterior, você terá oportunidade de praticar suas habilidades com algumas frases e sentenças.

JOYSTICKS

Sistema de controle dos mais versáteis e baratos, o joystick é um periférico que serve tanto para aplicações "sérias" como para tornar os jogos mais interessantes.

Embora estejam ficando cada vez mais sofisticados, os computadores domésticos ainda não conseguem dar instruções a si mesmos. Por enquanto (e esperamos que para sempre...), a responsabilidade disso recai inteiramente sobre o programador ou o usuário. Aprender a programar é, portanto, aprender a se comunicar com o computador.

O principal meio de comunicação do usuário com o computador continua sendo o teclado. Mas existem muitas razões para considerar esse meio como longe do ideal: a principal delas é que o teclado torna a comunicação comparativamente lenta em relação a outros métodos. Além disso, o uso eficiente do teclado exige que o usuário aprenda a digitar com rapidez e precisão.

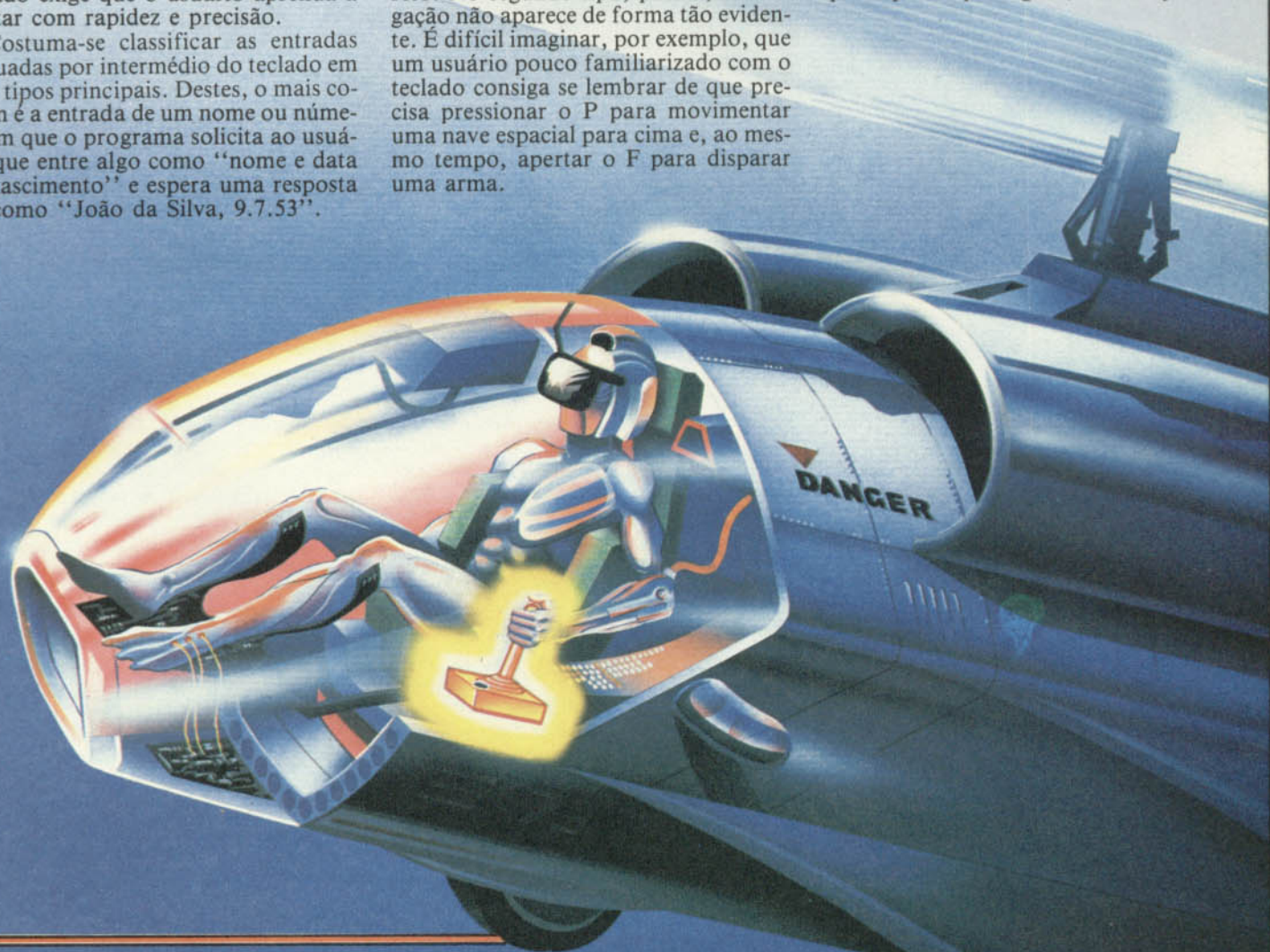
Costuma-se classificar as entradas efetuadas por intermédio do teclado em dois tipos principais. Destes, o mais comum é a entrada de um nome ou número em que o programa solicita ao usuário que entre algo como "nome e data de nascimento" e espera uma resposta tal como "João da Silva, 9.7.53".

O segundo tipo de entrada, por sua vez, transmite um tipo inteiramente diverso de informação. Nela, a pressão a uma tecla — em vez de provocar a entrada do caractere ao qual ela é normalmente ligada — aciona uma outra função bem diferente, determinada pelo programa. Por exemplo, se a tecla X for pressionada, o cursor da tela será movido para a direita; se for Z a tecla pressionada, o cursor será deslocado para a esquerda. Diversos capítulos do curso de *BASIC* e de *Programação de jogos* foram destinados a ensinar a programar esse tipo de entrada.

Obviamente, o primeiro tipo de entrada depende do teclado de modo direto. No segundo tipo, porém, essa ligação não aparece de forma tão evidente. É difícil imaginar, por exemplo, que um usuário pouco familiarizado com o teclado consiga se lembrar de que precisa pressionar o P para movimentar uma nave espacial para cima e, ao mesmo tempo, apertar o F para disparar uma arma.

Felizmente, existe uma alternativa: o joystick, um periférico barato e acessível para microcomputadores. Talvez injustamente associado apenas aos jogos, esse periférico cumpre diversas outras funções.

O joystick é uma espécie de alavanca que pode ser movida em várias direções pelo usuário. A cada movimentação, ele envia ao computador um sinal que indica a sua nova posição. A maioria dos joysticks tem um ou mais botões pulsantes (chamados de *botões de disparo*) que, ao serem pressionados, geram sinais diferentes para o computador (joystick é um termo derivado do aeromodelismo e não tem tradução adequada para o português; a mais aproxi-

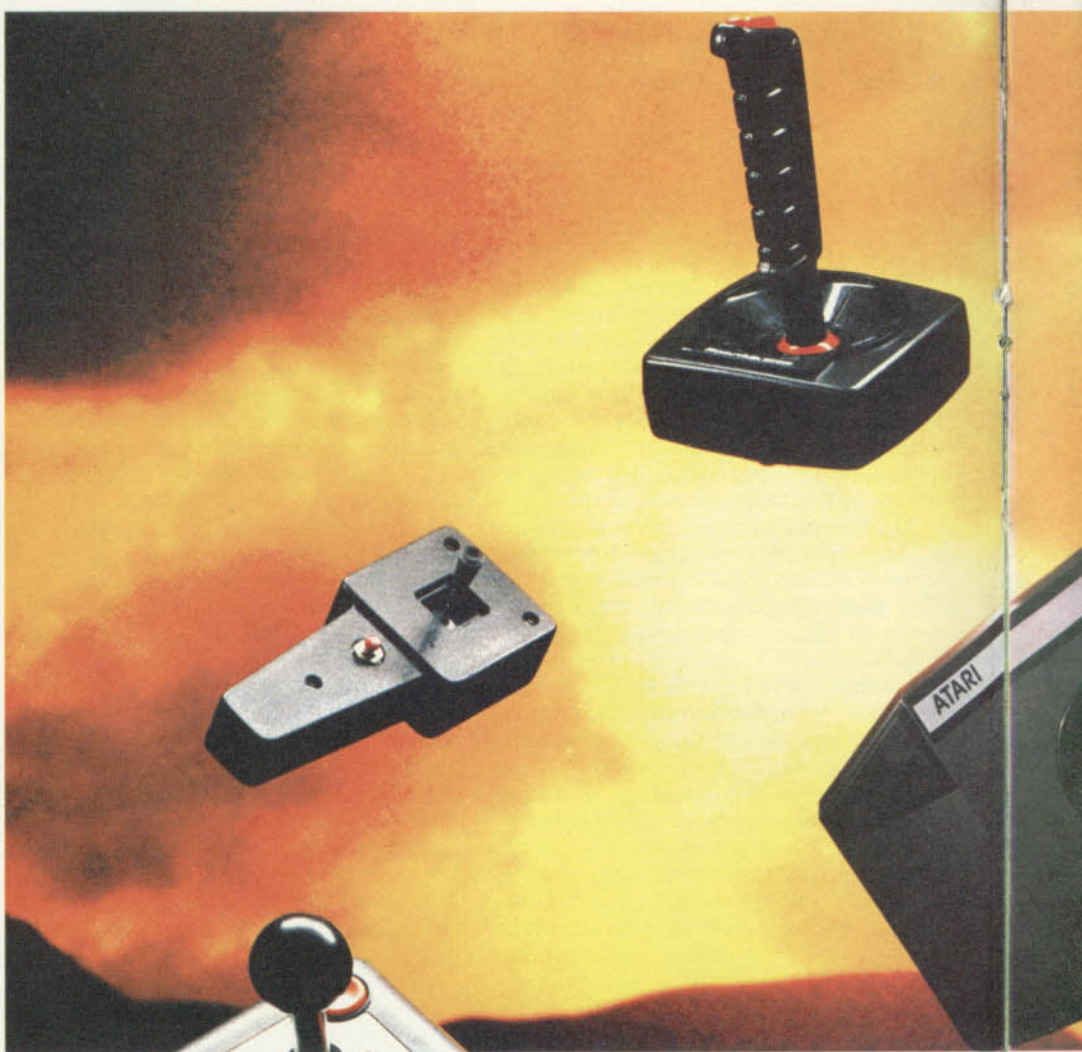
mada seria *manche*).

Com uma programação apropriada, mesmo nos jogos de ação mais modestos, os joysticks podem assumir várias das funções reservadas tradicionalmente para o teclado; assim, eles encontram muitas aplicações interessantes, úteis sobretudo para usuários que não querem ou têm dificuldades em usar o teclado.

Por exemplo, em um programa de gráficos, é perfeitamente possível utilizar um joystick ou um controle semelhante para guiar o cursor que desenhará na tela (na página 164 apresentamos um programa que faz isso por meio do teclado), ou para selecionar uma cor em uma "paleta" exibida na tela. Um joystick também pode ser usado para fornecer respostas alfabéticas ou numéricas. Alguns jogos de fliperama, por exemplo, possuem uma "lista de campeões", na qual são exibidos os nomes dos jogadores e seus recordes de pontos. Neste caso, uma entrada alfabética curta (por exemplo, as iniciais do jogador) é realizada, usando-se o joystick para mover o cursor até a posição onde está a letra desejada e, em seguida, o botão de disparo para selecioná-la. Com um pouco de imaginação, é fácil incorporar esse truque na digitação de nomes e números maiores ou, mais comumente, para efetuar uma escolha entre um menu de opções de programa, por exemplo.

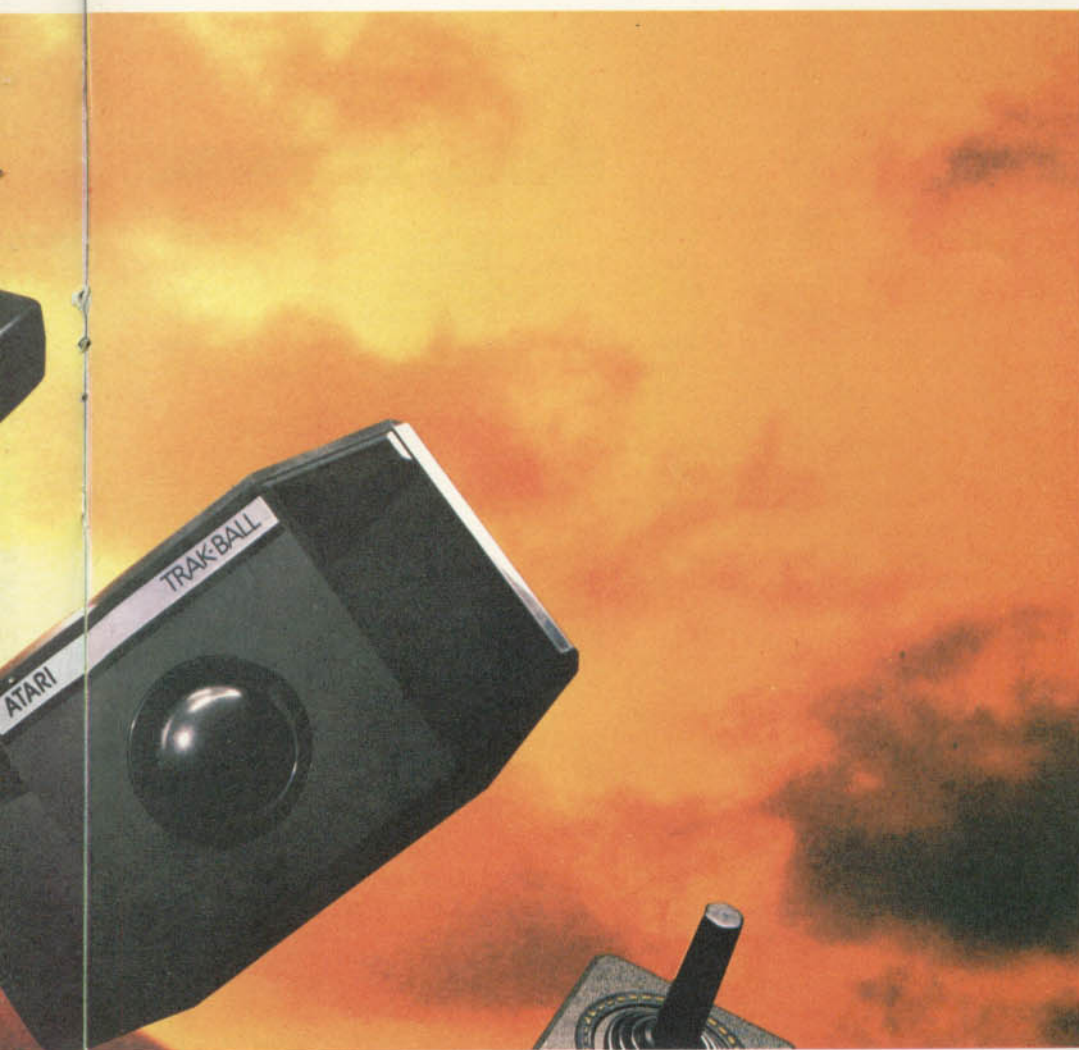
Em um artigo futuro, você verá como programar o seu computador para operar sob o controle de um joystick, de modo a tornar um programa mais atraente ou um videogame mais divertido. Antes disso, porém, é necessário entender quais são os diferentes tipos de joysticks existentes para computadores, e como podem ser utilizados.

TIPOS DE JOYSTICKS

Nem todos os joysticks são adequados aos vários tipos de computadores pessoais. Antes de adquiri-los, convém certificar-se de que eles são compatíveis com a sua máquina. Existe ainda uma outra restrição: ao se comprar um programa de jogos (ou outro qualquer) que utilize joysticks, é necessário verificar cuidadosamente se ele foi escrito para o tipo de joystick que você possui. Dentro dessas limitações básicas podem existir, contudo, diversas opções de compra.

O tipo mais simples de joystick consiste em uma caixa rasa com uma alavanca, que pode ser movimentada em oito direções diferentes: para a esquerda, para a direita, para cima, para baixo e para as quatro posições intermediárias, em diagonal. Além disso, ele conta com um botão de disparo que, apesar do nome, pode ser utilizado para tarefas como selecionar uma letra ou assinalar uma opção. Esse gênero de joystick é muito usado em videogames; por isso, é comumente chamado de "tipo Atari" ou digital.

Alguns desses joysticks possuem dois ou mais botões de disparo que podem ser úteis para jogadores canhotos, por exemplo, ou para aumentar a velocidade ou o grau de conforto na manipulação de jogos difíceis. Esses botões se localizam normalmente na base, mas podem situar-se também na ponta da alavanca de controle. Dependendo do programa, um botão pode, por exemplo, ser usado para disparar um canhão, enquanto o outro aciona um dispositivo que deixa cair uma bomba; um botão pode apagar uma linha de texto, enquanto o outro a edita, e assim por diante. Alguns modelos são especialmente planejados para se amoldarem à mão. Outros são modelados no formato de um cabo de pistola, complementado, em alguns casos, com um gatilho.

Joysticks mais modernos incluem dispositivos extremamente sofisticados, como ponteiros munidos de indicadores ultra-sensíveis de gravidade. Esses indicadores servem para detectar pequenos movimentos ou inclinações da mão do usuário (por exemplo, quando este aponta para alguma locação na tela). Eles utilizam interruptores de mercúrio: sempre que a alavanca é inclinada, o mercúrio em seu interior se desloca para uma das extremidades, fechando ou abrindo um contato elétrico. Tais dispositivos são tão sensíveis que não devem ser usados em jogos, a menos que o programa tenha sido escrito especialmente para eles.

Outro elemento da família dos joysticks é a "raquete eletrônica" (*paddle*), que nada mais é do que um botão giratório, tipo potenciômetro. Neste caso, a posição angular do potenciômetro é detectada pelo computador, que pode usar a informação para mover um cursor na tela, geralmente no sentido horizontal.

Alguns tipos de joysticks resultam de uma combinação de dois paddles, um na posição horizontal e outro na vertical.

Uma outra variante do joystick é a "trackerball" (esfera de comando). Originalmente desenvolvida para aplicações militares e "aviônicas", a esfera de comando foi logo adaptada para máquinas comerciais de jogos de fliperama e estão entrando agora no mercado dos computadores domésticos. Nesse periférico, o controle é feito por meio de uma bola que se projeta acima da superfície da caixa e que pode ser rolada em qualquer direção, com as pontas dos dedos.

Os modelos mais sofisticados desses dispositivos de controle são empregados também em computadores de aplicação comercial e profissional. Um dos mais recentes é o chamado "camundongo" (*mouse*), que se parece com uma trackerball de cabeça para baixo, embora existam também versões que funcionam de acordo com outros sistemas. O camundongo entra em ação quando se rola o dispositivo sobre a superfície de uma mesa. Seus movimentos são detectados por um programa especial, provocando deslocamentos correspondentes de um cursor gráfico ou de texto na tela. Assim, o camundongo pode substituir as teclas de controle do cursor. Ele permite movimentos rápidos sobre a tela para alterar dados ou para solucionar itens de um menu. As seleções são feitas por meio de um ou dois botões no alto da caixa. Ao contrário dos joysticks normais, o camundongo é utilizado para desenhar na tela, mas quase nunca para controlar movimentos em jogos.

Existem muitos outros tipos de dispositivos que servem para controlar o cursor na tela, mas cujo funcionamento é bem diferente do dos joysticks e de seus "parentes". Um deles é o *tablete gráfico*, que consiste em uma mesa retangular, sobre a qual se pode "escrever" com uma caneta especial, ou mesmo com o dedo (*touchpad*). Sensores localizados na superfície do tablete ou na pena transmitem ao computador as coordenadas X e Y da ponta da caneta.

As *canetas ópticas* são outro "parente" dos joysticks, que permitem o desenho direto sobre a tela, a escolha de opções, etc., e estão se tornando cada vez mais populares entre os usuários de computadores domésticos.

COMO FUNCIONAM

Embora pareçam diferentes à primeira vista, na verdade todos esses dispositivos funcionam de maneira semelhante. Em todos eles, o movimento é transformado em uma série de sinais elétricos que são "lidos" pelo computador.

Assim. como qualquer dispositivo eletrônico, os joysticks podem ser tanto digitais quanto analógicos. Normalmente, uma linha de computadores aceita apenas um dos tipos e mais raramente os dois. No joystick digital existem diversos comutadores eletrônicos que são abertos ou fechados conforme o ângulo de movimentação da alavanca: assim, um padrão único de bits é gerado para cada posição. Os paddles e joysticks de tipo analógico, por sua vez, funcionam de forma diferente: eles são construídos com um ou dois potenciômetros (resistores variáveis), cujo ângulo de rotação é proporcional ao deslocamento da alavanca nos sentidos horizontal e vertical. As variações de resistência obtidas com essas rotações são transmitidas ao computador na forma de duas voltagens proporcionais, de modo a fornecer uma única combinação para cada posição de controle.

Do ponto de vista visual, os dois tipos de joystick são também bastante diferentes. No tipo analogico, os potenciômetros são montados em ângulos retos e operados por um enlace mecânico. A alavanca, em conseqüência, não fica balanceada ao centro, ou seja, permanece na posição em que o usuário a deixar. Os tipos digitais, ao contrário, são normalmente balanceados ao centro e, quando liberados, voltam automaticamente para a posição central ou neutra. No entanto, alguns modelos analógicos são balanceados ao centro.

Por outro lado, uma nítida diferença no comportamento mecânico separa os dois sistemas. Os joysticks digitais (balanceados ao centro) são mais duros e normalmente só se movimentam por uma pequena distância. Assim, um esforço maior feito pelo jogador é contrabalançado por uma maior resistência mecânica.

Na verdade, o efeito que ambos os ti-

pos provocam no computador é controlado mais pelo programa do que pelo próprio joystick. Geralmente, um programa bem escrito torna o joystick mais sensível e fácil de controlar. Por exemplo, um programa pode determinar que um leve movimento da alavanca seja suficiente para dar início a mudanças de direção de um objeto na tela, sem que haja necessidade de se pressionar continuamente a alavanca.

Já os tabletes digitalizadores que funcionam por toque (*touchpads*) consistem em um sanduíche de duas folhas de plástico, separadas por uma pequena distância. Uma fina grade de resistores ou fios condutores é construída sobre uma das faces dessas folhas, orientadas de modo a formar ângulos retos entre si. Ao menor contato de um dedo ou de um lápis com o tablete, os condutores situados nas duas folhas se tocam e geram um padrão de voltagens, que é varrido pelo computador. Os touchpads, entretanto, podem apresentar problemas, devido à dificuldade de se conseguir uma superfície uniformemente sensível. Uma desvantagem em utilizá-los é que, se a superfície se sujar — por marcas de dedos, por exemplo —, os movimentos tenderão a se tornar erráticos devido ao deslizamento do dedo pela superfície.

Extremamente leves, as trackerballs podem ser operadas com facilidade, pois a bola não tem conexão mecânica direta com o sistema sensor. Ela é sustentada sobre rolamentos que giram livremente em duas direções, em ângulo reto uma em relação à outra. Quando a bola sofre uma rotação, ela faz rodar um ou ambos os rolamentos. Estes, por sua vez, estão conectados a potenciômetros ou a um sensor digital — um sistema formado por um disco rotatório para interromper um raio de luz proveniente de um diodo LED (foto-emissor), que cai sobre um fototransistor (fotosensor). Com a contagem de impulsos de interrupção, o computador fica "sabendo" de quanto girou o rolamento. Mais uma vez, qualquer que seja o sistema utilizado, o computador é capaz de interpretar os sinais elétricos em termos de um padrão determinado de movimentação na tela.

OPERAÇÃO COM O COMPUTADOR

Os comandos **GET$** ou **INKEY$** do BASIC são os mais utilizados para programar a execução de funções de movimentação do cursor de vídeo, sob controle de determinadas teclas do teclado. Eles efetuam uma "varredura" do teclado, de modo a assinalar ao programa se alguma tecla foi pressionada. Se isso acontecer, o programa poderá efetuar alguma operação específica — tal como a movimentação de uma base de mísseis para a direita, se a tecla X, por exemplo, for pressionada.

A operação do joystick digital não difere essencialmente disso. Mas enquanto o teclado é parte integrante do computador, o joystick é um elemento externo acoplado a ele. Assim, o joystick deve primeiro ser conectado ao computador através de uma porta adequada (um conector especial); é por essa porta que o micro normalmente se comunica com o mundo exterior. Em seguida, é necessário colocar na memória RAM do micro um programa adequado para "varrer" periodicamente a porta de entrada, de modo a procurar por um sinal específico, que signifique que o joystick está sendo movimentado.

Alguns problemas de compatibilidade podem surgir entre esses periféricos e o computador. Em primeiro lugar, o joystick precisa ser conectável à porta de entrada do micro, seja diretamente, seja através de algum tipo de interface. Além disso é necessário que o programador conheça os sinais gerados pelo joystick; caso contrário, não será possível programar o computador para procurá-los.

Essas exigências dão lugar, às vezes, a tremendas confusões. Felizmente, já existem padrões de mercado seguidos em maior ou menor grau por diferentes fabricantes. No Brasil, cada linha de microcomputadores apresenta um conjunto bem definido de convenções. O padrão mais comum, que se tornou virtualmente universal, é o do joystick tipo Atari (digital e centro-balanceado). Existem diversos fabricantes de joysticks desse tipo, e alguns computadores são projetados de modo a aceitá-los diretamente.

É o caso, por exemplo, da linha Sinclair (ZX-81 e Spectrum), cujos compatíveis nacionais (TK-85, TK-90X, etc.) incluem um conector para joysticks tipo Atari. Os modelos da linha MSX têm, por sua vez, entrada para dois joysticks tipo Atari. Os compatíveis com o TRS-Color (por exemplo, o Prológica CP-400) utilizam joysticks analógicos. Já os micros da linha TRS-80 normalmente não contam com nenhum tipo de entrada para joystick.

Além disso, podem existir, às vezes, diferenças importantes entre diversos modelos de máquinas da mesma linha, como no caso do Apple, que tem protótipos nacionais que aceitam joysticks digitais tipo Atari (é o caso do TK-2000), ou analógicos (*paddles*).

Uma das diferenças mais importantes entre micros diz respeito ao interfaceamento.

INTERFACEAMENTO

Como qualquer periférico de computador, os joysticks precisam ser conectados à UCP por intermédio de uma interface adequada. A interface pode estar integrada à configuração básica da UCP do computador, ou pode ser um dispositivo separado.

Muitos fabricantes incorporam aos seus micros um conector e uma interface do tipo Atari, de modo a fazê-los aceitar qualquer joystick que seja de um modelo compatível com esse padrão.

Outra saída seria dispor de uma ou mais portas analógicas, que aceitam paddles ou joysticks do tipo potenciômetro. No entanto, são raros os micros que já incluem essas portas em sua configuração básica, pois isto significa que devem ser construídos dois conversores analógico-digitais para cada joystick a ser utilizado.

Os micros da linha Sinclair Spectrum não vêm acompanhados de joysticks. Assim, para ajustar a eles um desses periféricos, é necessário uma interface apropriada ao conector de borda existente atrás do console.

A interface é alojada em uma caixa separada que se situa na parte posterior da máquina, em contato direto com os terminais do conector de borda, na entrada do usuário. Existe, porém, um tipo de interface (disponível apenas no Exterior) construída dentro da caixa do próprio joystick, com um conector separado para o cabo de ligação.

O joystick mais adequado para o Spectrum é compatível com o tipo Atari. No Exterior, entretanto, existem muitos outros tipos de joysticks, que são incompatíveis entre si no que se refere aos sinais que geram no conector de expansão.

Por isso, o usuário do Spectrum deve tomar muito cuidado quando comprar ou copiar programas provenientes do Exterior que utilizem joysticks, pois eles nem sempre funcionam com o joystick disponível no Brasil. Assim, é conveniente verificar a documentação do software antes de adquiri-lo. Alguns programas mais sofisticados incluem um menu de opções que permitem ao usuário selecionar o tipo de joystick a ser usado para acionar o jogo.

Outro problema para os proprietários

do Spectrum é que as interfaces mais comuns desse micro só possibilitam a conexão de um joystick. Entretanto, algumas interfaces mais sofisticadas permitem a conexão de dois joysticks ao mesmo tempo. Infelizmente, porém, são raros os softwares compatíveis com tais interfaces.

O interpretador BASIC do Spectrum não tem nenhum comando específico para programação com joysticks. Assim, o mais comum é usar o comando **INKEY$**.

O ZX-81 está longe de ser uma máquina ideal para jogos. Entretanto, assim como o Spectrum, é possível conectá-lo a um joystick do tipo Atari por intermédio de um conector Phillips, situado na lateral ou na traseira do console. Portanto, a operação com esse micro é parecida com as descritas acima.

O BASIC do ZX-81 não tem nenhum comando específico para a programação de joysticks, como é o caso dos micros da linha MSX, Apple e TRS-Color, mas o comando **INKEY$** pode ser usado da mesma forma que com o teclado, pois os joysticks para o ZX-81 geram sinais correspondentes às teclas de controle de cursor (6, 7, 8 e 9, combinadas com <**SHIFT**>).

Isto é uma vantagem adicional, pois um jogo que utilize tais teclas operará igualmente bem, com ou sem joystick (só que ficará mais divertido com o joystick, como é o caso de um popular programa de simulação de vôo de uma aeronave).

Os microcomputadores da linha TRS-Color incluem duas entradas para joysticks analógicos, do tipo potenciômetro de centro não-balanceado. Esses joysticks têm botões de disparo, e podem ser facilmente programados por comandos específicos do interpretador BASIC. A tela de alta resolução gráfica, quando associada por programação ao uso desses joysticks, permite efeitos sensacionais de animação gráfica, em jogos ou no desenho livre.

Os micros da linha TRS-80 não incluem nenhuma previsão ou porta especial para joysticks.

Entretanto, esses computadores podem ser conectados tanto a joysticks digitais do tipo Atari, quanto a joysticks e paddles analógicos; para isso, existem produtos comercialmente disponíveis no mercado.

No primeiro caso, é necessário efetuar uma modificação no hardware, conectando fios em paralelo com a interface do teclado. Em seguida, deve-se ligar a máquina ao joystick. Uma vez em operação, este gera sinais semelhantes aos provocados pela pressão das teclas de controle do cursor, que podem ser detectadas por meio do comando **INKEY$**, do BASIC. Há um dispositivo fabricado no Brasil — "joypad", um pequeno teclado paralelo apenas com as teclas de controle de cursor (*joypad*) — cuja função consiste em facilitar o desempenho em jogos.

Para conectar dispositivos analógicos, é necessário adquirir uma interface especial de conversão analógico-digital, que é ligada à porta de expansão, na parte de trás da UCP.

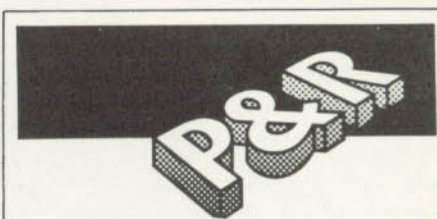
No TRS-80 não existem comandos específicos do BASIC para os dois tipos de joysticks.

Os microcomputadores da linha MSX incluem portas de entrada para joysticks do tipo digital centro-balanceado, com dois ou mais botões de disparo. No Brasil, cada fabricante oferece o seu modelo próprio de joystick, mas o padrão de conexão e geração de sinais segue o do Atari.

Da mesma forma que o TRS-Color, o interpretador BASIC dessas máquinas contém comandos bastante poderosos.

A maioria dos micros da linha Apple apresenta no console duas portas para dois paddles analógicos, ou para um joystick de dois eixos (não centro-balanceado). O interpretador BASIC incorpora comandos para leitura da posição dos potenciômetros e da pressão ao botão de disparo.

Existem ainda no mercado joysticks do tipo digital, que podem ser conectados via porta serial ou paralela.

O Microdigital TK-2000 inclui, na versão padrão, um conector específico para joystick digital do tipo Atari, que opera paralelamente ao teclado normal (ou seja, gera sinais correspondentes às teclas de controle do cursor e às teclas **FIRE** dos dois lados do teclado). Assim, a utilização do joystick pode ser programada por meio de comandos **GET$** ou **PEEK** do teclado existentes no BASIC do TK-2000. Não existem joysticks ou paddles do tipo analógico para esse computador.

Como funciona um conversor analógico-digital?

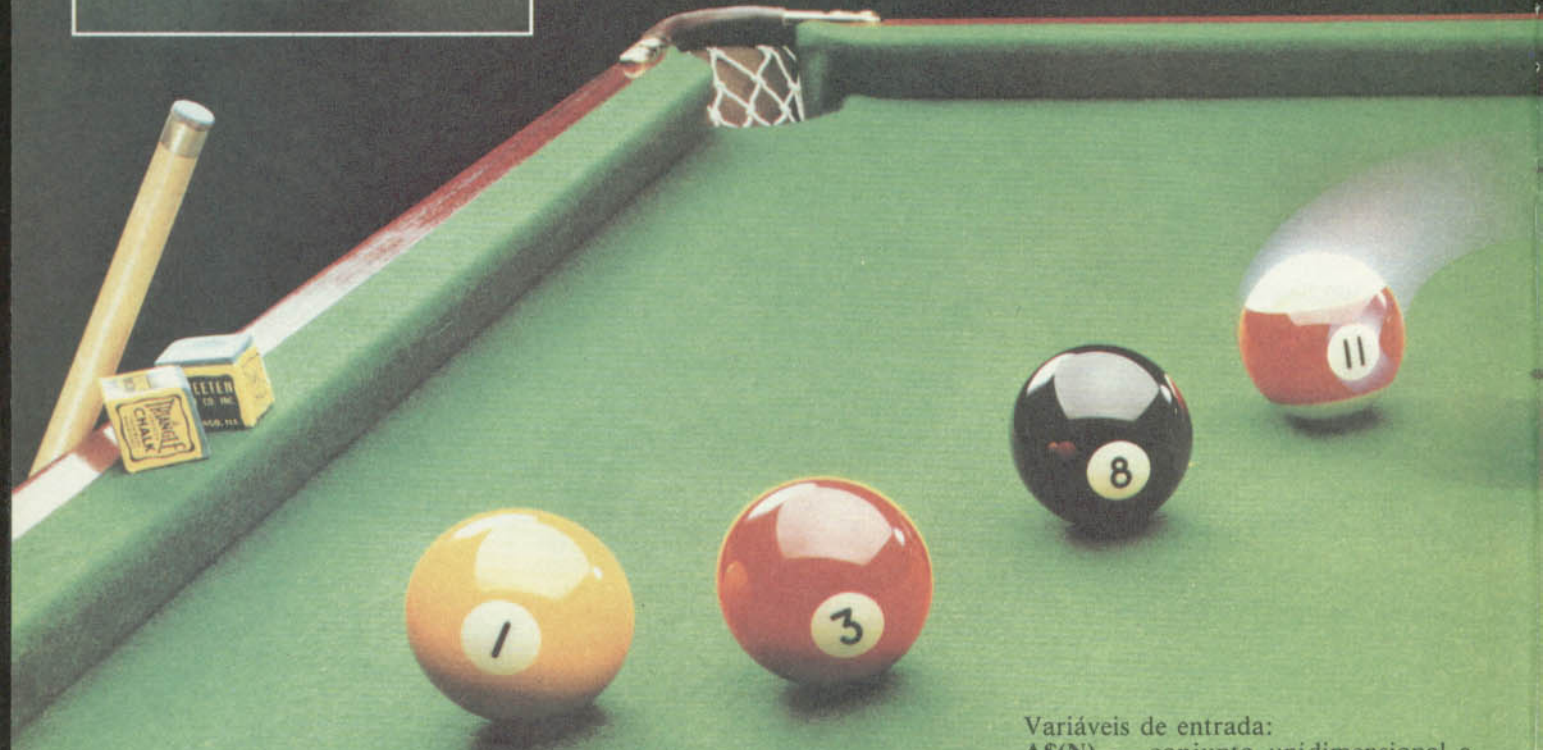
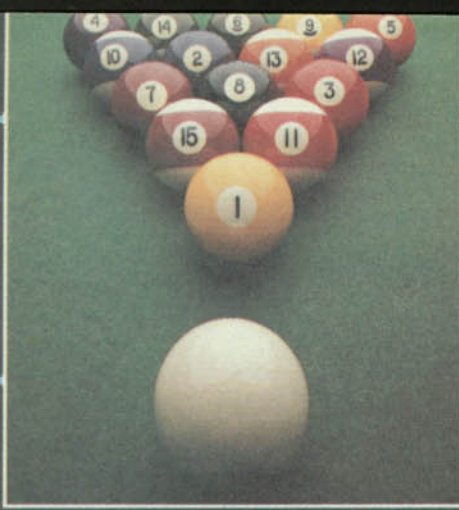
Necessário para converter em números binários as voltagens contínuas geradas pelos paddles e joysticks analógicos, o conversor A/D realiza uma tarefa de *digitalização* do sinal elétrico de entrada.

Suponhamos que uma rotação de 118 graus num dos potenciômetros do joystick gera uma voltagem proporcional entre 0 e 5 volts. O conversor A/D tem um comparador interno que gera voltagens sucessivas, divididas em um certo número de intervalos (por exemplo, 256 intervalos de 0.0195 volts cada, se for um conversor de oito bits). A cada nível interno de voltagem, o circuito compara-o com o nível externo de voltagem (gerado pelo joystick) e decide se esses valores são aproximadamente iguais ou não. Ao mesmo tempo, outro circuito digital vai contando quantos desses níveis foram testados. Assim que o comparador informar que a igualdade foi atingida, o conversor A/D enviará ao computador, através da interface paralela, o byte contendo o número de contagens. Por exemplo, se a voltagem gerada pelo joystick for 1.95 volts, o número enviado para o computador será 100!

CUIDADO AO COMPRAR UM JOYSTICK!

Ao comprar um joystick, verifique primeiro se o modelo escolhido é compatível com o seu computador e, no caso do Spectrum, com o seu software também.

Em seguida, pense em quanto você está disposto a gastar; se você é proprietário de um Spectrum, tenha em mente que a interface poderá custar tanto quanto o joystick, ou até mais. Os tipos mais simples de joysticks custam o mesmo que um programa de jogo, enquanto os mais elaborados podem ser duas ou três vezes mais dispendiosos.

ORDENAÇÃO PELO MÉTODO DE BOLHAS

Uma vez elaborado o projeto geral de um programa e escritos os módulos individuais, pode-se começar a pensar em testar os módulos. Em seguida, é necessário planejar a forma de colocá-los todos juntos.

As sub-rotinas ou módulos precisam ser colocados de alguma forma dentro do contexto do programa. Normalmente, a ligação é feita por meio de variáveis. Algumas destas — conhecidas como *parâmetros de entrada* — são especificadas no início, e passadas à rotina. Outras variáveis retornam ao programa por intermédio da rotina; estas são os *parâmetros de saída*. É muito importante que as variáveis sejam especificadas de um modo preciso para não serem confundidas umas com as outras.

Ao se começar a escrever um programa deve-se fazer uma lista de todas as variáveis necessárias, juntamente com uma descrição dos seus usos e possíveis valores iniciais, caso sejam conhecidos. Do contrário, mesmo que se saiba no início o que significam todas as letras, corre-se o risco de esquecer de retornar ao programa mais tarde. Um recurso bastante prático, caso o computador permita ou o espaço de memória não seja muito pequeno, consiste em adotar nomes longos para as variáveis (veja na página 99 uma tabela do que é permitido para cada linha de microcomputadores).

ROTINA DE ORDENAÇÃO TIPO BOLHA

Como se deve especificar as variáveis para uma rotina de ordenação tipo bolha? O exemplo a seguir apresenta uma porção específica de matriz ordenada alfabeticamente.

Variáveis de entrada:
A$(N) — conjunto unidimensional a ser classificado (tamanho de **N > = 1**)
N1 — primeiro item em conjunto a ser classificado (**1 < = N1 < = N2**)
N2 — último item em conjunto a ser classificado (**N1 < = N2 <** = o tamanho de **A**)

Variáveis de saída:
A$(N) — conjunto ordenado.

Variáveis temporárias:
Z, Z$, I.

Uma providência muito útil para evitar conflitos entre módulos ou com o resto do programa consiste em listar as variáveis temporárias utilizadas em uma sub-rotina. Também é útil reservar algumas letras, especialmente para variáveis temporárias: por exemplo, as variáveis de **Z0** a **Z9**. Esse tipo de recurso evita desperdícios de espaço das variáveis.

Alguns erros ou defeitos de programa são causados às vezes por variáveis *viciadas* — isto é, com valores mudados.

Aperfeiçoe as técnicas da programação estruturada, aprendendo a construir um programa de ordenação pelo método de bolhas, e coloque em ordem o que quiser.

- DEFINA AS VARIÁVEIS
- COMO ESCREVER UMA SUB-ROTINA DE ORDENAÇÃO
- COMO TESTAR OS MÓDULOS
- COLOQUE TUDO JUNTO

Esses problemas, contudo, podem ser evitados com a aplicação do método exposto acima.

Uma listagem de variáveis também é útil para o caso de o programa vir a ser modificado mais tarde. Essa medida assegura maior rapidez na alteração das variáveis, impedindo ao mesmo tempo o aparecimento de erros extras no programa. Tais erros são normalmente provocados por variáveis viciadas, utilizadas para outros propósitos. Lembre-se ainda de anotar as modificações, juntamente com as datas em que foram feitas.

Eis aqui o programa para uma rotina de ordenação por bolhas (seu funcionamento será explicado mais adiante, na lição número 23 de *Programação BASIC*).

```
1000 REM ORDENAÇAO TIPO BOLHA (
AS(N),N1,N2)
1010 LET Z=0
1020 FOR I=N1 TO N2-1
1030 IF A$(I)<=A$(I+1) THEN GOT
O 1080
1040 LET Z$=A$(I)
1050 LET A$(I)=A$(I+1)
1060 LET A$(I+1)=Z$
1070 LET Z=1
1080 NEXT I
1090 IF Z=1 THEN GOTO 1010
1100 RETURN
```

Como toda sub-rotina, esta deve ser chamada a partir do programa principal. Por exemplo, para ordenar os itens de 5 a 20, a rotina pode ser chamada assim:

```
100 LET N1=5:LET N2=20:GOSUB
1000: REM ORDENACAO TIPO BOLHA
```

É preciso também que exista uma seção do programa que lhe permita entrar os itens a serem ordenados, e outra que imprima a lista ordenada. Neste estágio você deve decidir como quer que a exibição apareça na tela.

Se você trabalha com um MSX, pode tirar proveito do poderoso BASIC desse micro. Mude a linha 1040 para **SWAP A$(1),A$(I + 1)** e elimine as linhas 1050 e 1060. Seu computador trocará o conteúdo das duas variáveis de uma só vez, dispensando o uso da variável **Z$**.

COMO TESTAR OS MÓDULOS

Cada módulo do projeto original pode se tornar uma sub-rotina no seu programa. Esse método de dividir o programa é muito útil durante o estágio de teste, pois os módulos podem ser postos à prova ou depurados individualmente. Volte agora à sub-rotina da ordenação tipo bolha, testando-a assim:

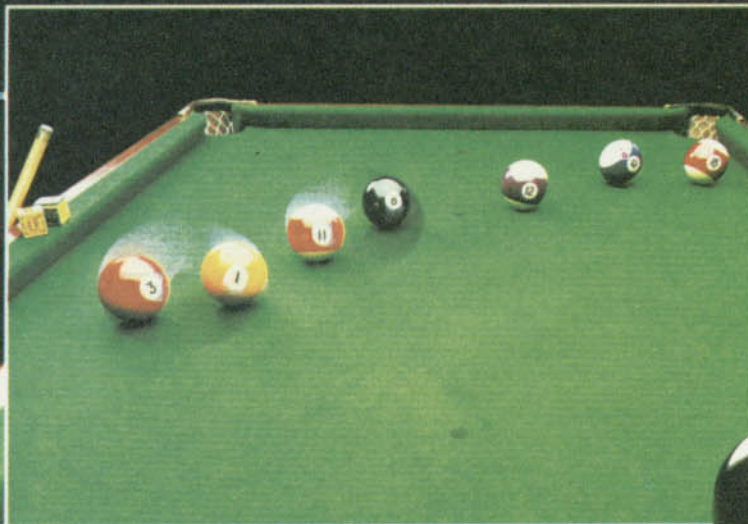
```
8 INPUT "Número de ítens ";N
10 DIM A$(N)
12 PRINT"Entrada dos ítens da m
atriz:"
14 FOR I=1 TO N: INPUT A$(I):NE
XT I
16 INPUT "Intervalo a ser orden
ado ";N1,N2
18 GOSUB 1000
20 PRINT"Lista ordenada:"
22 FOR I=1 TO N:PRINT A$(I):NEX
T I
24 GOTO 16
```

O programa acima também funcionará no TRS-Color e TRS-80, desde que se adicione a linha a seguir, cujo objetivo é reservar um espaço suficiente de memória:

```
6 CLEAR 1000
```

No ZX-81 e no Spectrum, modifique a linha 10 do programa acima para:

```
10 DIM A$(N,10)
```

No ZX-81, fracione as linhas com declarações múltiplas, mude tudo para maiúsculas, e altere a linha 8 para:

```
8 PRINT "NUMERO DE ITENS"
9 INPUT N
```

Entretanto, a tarefa de testar cada caso de entrada e de saída pode se revelar impossível em programas complexos, exigindo um tempo excessivamente grande. Apesar disso, os casos limítrofes podem ser checados. Por exemplo, a rotina seguinte pode ser verificada para entrar valores de 1,99 e 100.

```
1010 PRINT "ENTRE UM NUMERO
     (1-99) ";
1015 INPUT N
1020 IF N<1 OR N>99 THEN GOTO
     1010
1030 RETURN
```

Outra providência aconselhável é certificar-se de que cada linha do programa foi rodada pelo menos uma vez durante o estágio de depuração. Assim, todos os desvios condicionais, tal como a declaração **IF**, poderão ser chocados com ambas as condições de verdadeiro ou falso.

COLOQUE TUDO JUNTO

Finalmente, todos os módulos poderão ser encadeados e o programa testado como um todo. Isso é conhecido como *integração do programa*. Se ocorreram problemas, qualquer módulo suspeito poderá ser checado novamente e modificado em caso de necessidade.

Cumpridas todas essas exigências, você terá um programa perfeitamente estruturado que fará exatamente o que for determinado.

As regras para se escrever um programa estruturado são, resumidamente, as seguintes:

1. Escreva uma descrição geral do programa.
2. Divida-a em tantos módulos quantos forem os níveis necessários.
3. Desenhe um fluxograma para cada módulo e defina as variáveis de entrada e de saída e de quaisquer outros efeitos, tais como a exibição na tela.
4. Escreva os programas para cada módulo que utilizar as estruturas descritas na parte 1.
5. Teste os módulos, fornecendo as entradas e checando os resultados e as saídas.
6. Combine todos os módulos e teste o conjunto do programa.

A finalidade de todo esse trabalho é aumentar a legibilidade de um programa, assim como sua capacidade, segurança e transportabilidade. Além disso, ele facilita a tarefa de testar e modificar o programa.

COMO FUNCIONA O MÉTODO DAS BOLHAS

O programa de ordenação pelo método das bolhas foi utilizado aqui para mostrar como um módulo pode ser construído e depois testado, antes de ser encadeado ao programa principal. As rotinas de ordenação são muito úteis para todos os tipos de programas. A que apresentamos agora classifica palavras em ordem alfabética, mas poderia servir também para classificar números em ordem crescente. Basta modificar as variáveis **Z$** para **Z**, e o conjunto **A$( )** para **A( )**.

O computador percorre a lista, comparando pares de itens, um de cada vez. Se eles estiverem em ordem correta não serão mudados. Se estiverem em ordem incorreta serão intercambiados. O programa continua através da lista, efetuando mais trocas até que todos os itens estejam em ordem correta.

Para avaliar o funcionamento do programa em detalhes é conveniente compará-lo com o fluxograma. A primeira parte do programa (que não está no diagrama) define o número de itens da lista — **N** — e estabelece um conjunto chamado **A$( )** com espaço suficiente para **N** itens. As linhas 12 e 14 pedem ao usuário as palavras que serão armazenadas no conjunto e a linha 16 pergunta quais delas serão ordenadas. Se você quiser ordenar a lista inteira, digite 1 seguido de uma vírgula, mais o valor de **N**. A sub-rotina é chamada na linha 18.

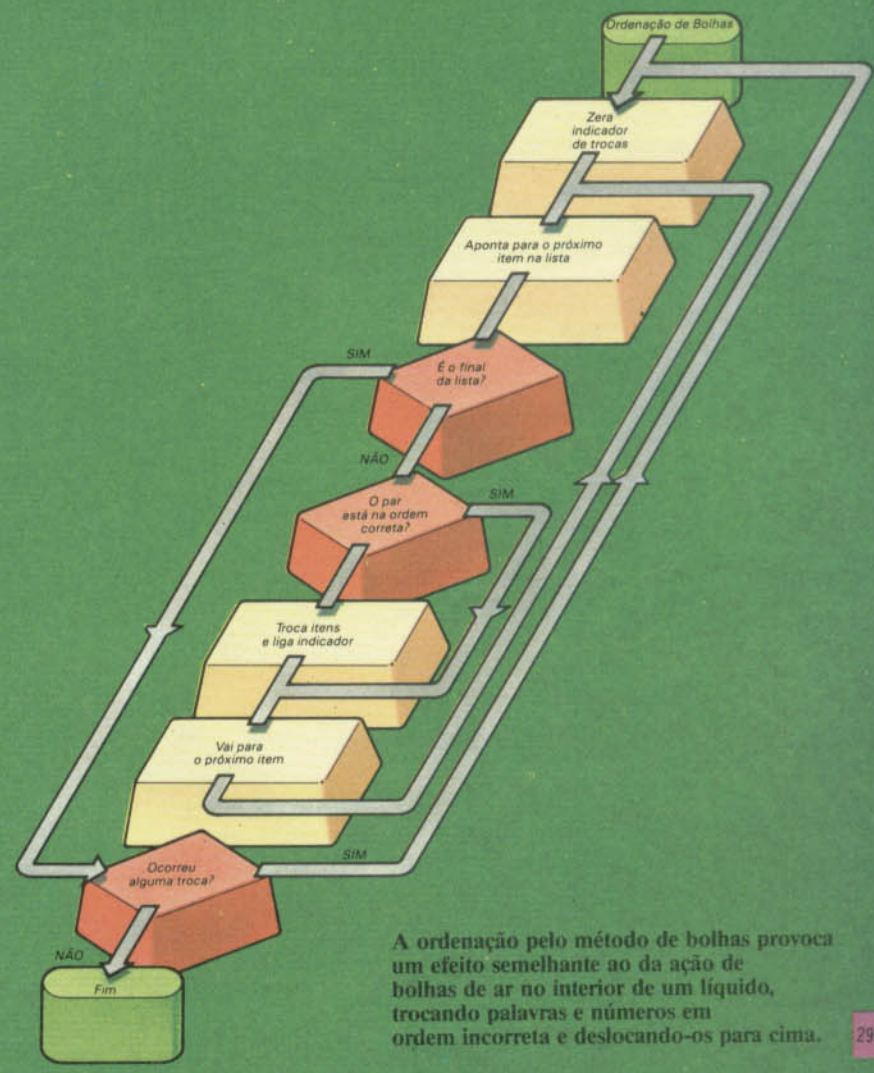
A rotina começa estabelecendo Z igual a 0. Z é conhecido como um *sina-*

lizador e registra se alguma troca de posições foi realizada.

A linha 1020 cria um laço para percorrer a lista. Os números certificam de que cada par foi comparado pelo menos uma vez. A linha 1030 compara as primeiras duas palavras; se estas estiverem em ordem correta, o programa pulará a rotina de trocas e se dirigirá para o par seguinte.

As linhas 1040 a 1070 somente serão atingidas se as palavras estiverem em ordem incorreta. A primeira palavra é colocada em uma variável temporária **Z$**. A segunda é movimentada para alguma posição, e então a primeira palavra é deslocada uma posição abaixo, dentro do conjunto. Z é então igualado a 1 para mostrar que foi efetuada uma troca. A linha 1080 manda o computador de volta para comparar o par de palavras seguinte.

Uma vez que todos os pares tenham sido comparados, o programa atingirá a linha 1090. Se Z = 1, isso significa que pelo menos uma troca foi efetuada; dessa forma, ele percorrerá a lista novamente. Se nenhuma troca foi efetuada, então a lista está na ordem correta, a sub-rotina termina e as linhas 20 e 22 imprimirão a lista ordenada.

A denominação de método das bolhas tem, assim, sua razão de ser: o movimento das palavras lembra globos de ar em ascensão no interior de um líquido.

A ordenação pelo método de bolhas provoca um efeito semelhante ao da ação de bolhas de ar no interior de um líquido, trocando palavras e números em ordem incorreta e deslocando-os para cima.

ASSEMBLER PARA O TRS-COLOR

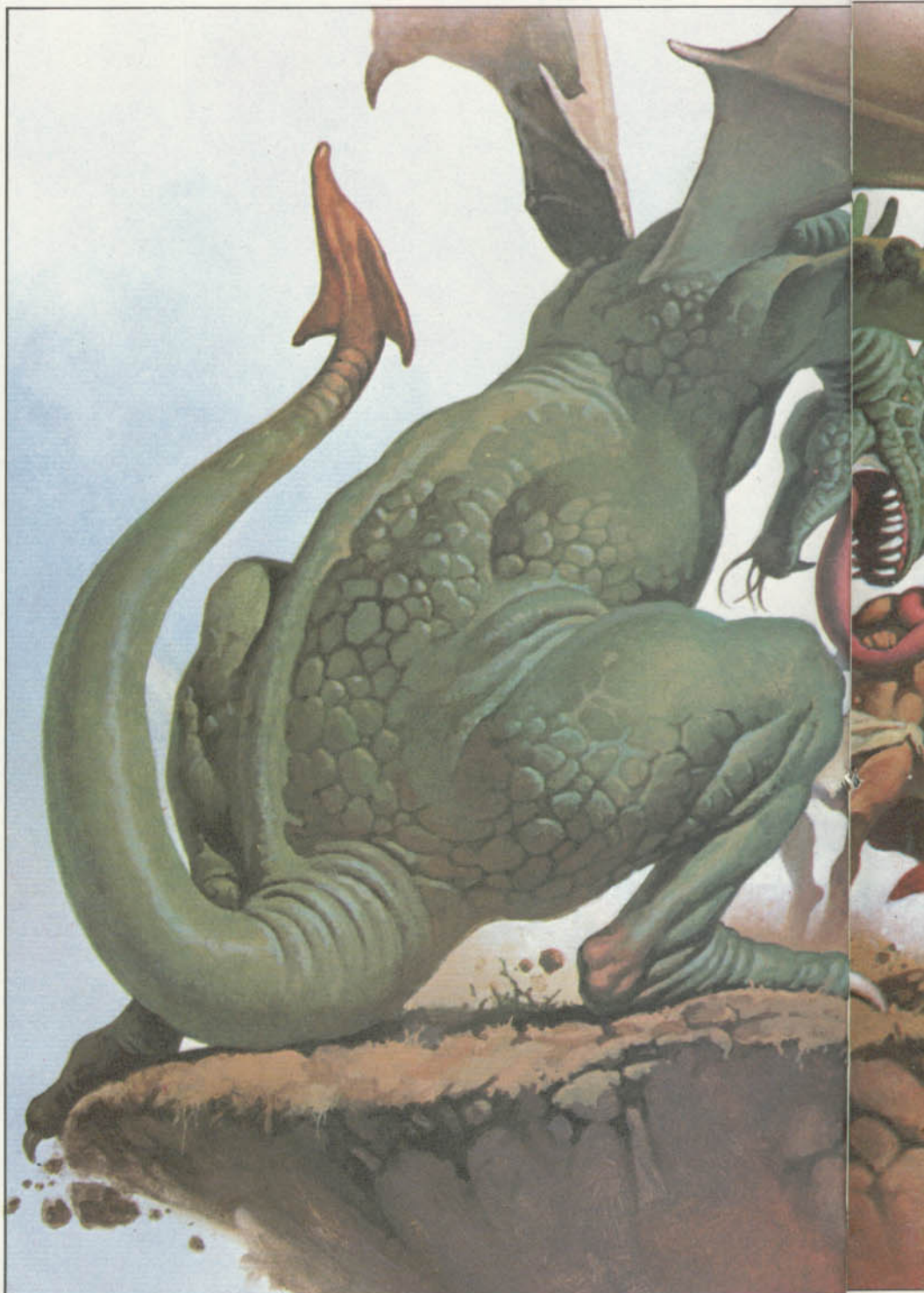
Bem mais rápido do que o programa para o Spectrum, o Assembler para os micros da linha TRS-Color proporciona igual eficiência. Economize tempo, aprendendo a trabalhar com ele.

Nesta lição, apresentamos um Assembler para o TRS-Color. Mais longo do que o programa para o Sinclair ZX Spectrum objeto da lição anterior, ele conta com a participação de um editor.

O microprocessador utilizado pelo TRS-Color é o Motorola 6809 de oito bits, cujo conjunto de instruções é bem menor que o do Z80, empregado pelo Spectrum. Entretanto, o Assembler do TRS-Color tem que passar por até três "etapas" para traduzir um comando de dezesseis bits, enquanto o programa do Spectrum deve passar por no máximo duas. Apesar dessa etapa adicional, o programa do TRS-Color é quase três vezes mais rápido que o do Spectrum: seu grande recurso para procurar os mnemônicos — a função **INSTR** — não existe no BASIC do Spectrum.

A linha 10 pode provocar um erro FC quando o programa é rodado pela primeira vez. Não se importe com a mensagem de erro e rode o programa novamente, que ele funcionará.

O ASSEMBLER

```
10 PMODE 0:PCLEAR 1:CLEAR 3000:
CLS:PRINT @233,"INICIALIZANDO":
R$=CHR$(13):POKE 146,1
20 DIM SK$(1),K1(94),K2(94),T$(
200),RR(100),Z$(100)
30 FOR CC=1 TO 94:READ K$,K1(CC
),K2(CC):C=CC/49:SK$(C)=SK$(C)+
RIGHT$(STR$(CC),2)+K$:NEXT
40 DATA ADCA,185,1,ADDA,187,1,A
DDD,243,2,ASL,120,3,CLR,127,3,C
MPA,177,1,CMPD,4275,2,CMPY,4284
,2,BCC,36,4,BCS,37,4,BEQ,39,4
50 DATA BHS,36,4,BLO,37,4,BMI,4
3,4,BNE,38,4,BPL,42,4,BRA,32,4,
LBRA,22,5,BSR,141,4,LBSR,23,5,C
MPX,188,2,CMPU,4531,2,CMPS,4540
,2,DEC,122,3
60 DATA INC,124,3,JSR,189,3,LDA
,182,1,LDB,246,1,LDD,252,2,LDS,
4350,3,LDU,254,3,LDX,190,3,LDY,
4286,3,LSL,120,3,LSR,116,3
70 DATA PSHS,52,1,PSHU,54,1,PUL
S,53,1,PULU,55,1,ROL,121,3,ROR,
118,3,RTS,57,,STA,183,3,STB,247
,3,STD,253,3,STS,4351,3,STU,255
```

- CONVERSÃO AUTOMÁTICA DA LINGUAGEM ASSEMBLY PARA CÓDIGO DE MÁQUINA
- COMO CALCULAR SALTOS E DESVIOS
- MANIPULAÇÃO DE PÓS-BYTES
- USO DE RÓTULOS
- COMO COLOCAR CÓDIGO DE MÁQUINA NA MEMÓRIA
- AUMENTA A VELOCIDADE

```
,3
80 DATA STX,191,3,STY,4287,3,SU
BA,176,1,SUBD,179,1,ANDA,180,1,
ABX,58,,ANDCC,76,1,ASR,119,3,RT
I,59,,SBCA,178,1,NOP,18,,NEG,11
2,3
90 DATA BITA,181,1,BGE,44,4,BGT
,46,4,BHI,34,4,BLE,47,4,BLS,35,
4,BLT,45,4,BRN,33,4,BVC,40,4,BV
S,41,4,EXG,30,1,TFR,31,1
100 DATA COM,115,3,CWAI,108,1,D
AA,25,,EORA,184,1,TST,125,3,LEA
S,66,3,LEAU,67,3,LEAX,64,3,LEAY
,65,3,MUL,61,,ORA,138,1,ORB,202
,1
110 DATA ORCC,74,1,SEX,29,,SWI,
63,,SWI2,4159,,SWI3,4415,,SYNC,
19,,EQU,-1,2,FCB,-2,1,FDB,-3,2,
RMB,-4,,JMP,126,3
120 DIM X$(13),V(14),KK(13),Y$(
13)
130 FOR C=0 TO 12:READ X$(C),V(
C),KK(C),Y$(C):NEXT
140 DATA PCR,253,7,,PC,253,8,D,
--,243,1,X,-,242,1,Y,X,159,,U,Y
,191,,S,U,223,,PC
150 DATA S,255,,,++,241,1,,+,24
0,1,A,A,246,1,B,B,245,1,CC,D,25
1,1,DP
160 FOR J=0 TO 9:READ PU$(J),PU
(J):NEXT
170 DATA PC,128,U,64,S,64,Y,32,
X,16,DP,8,D,6,B,4,A,2,CC,1
180 CLS:PRINT @43,"ASSEMBLER"R$
R$TAB(8)"G=LER DO GRAVADOR"R$S$
TAB(8)"S=SALVAR NO GRAVADOR"R$R
$TAB(8)"A=MONTAR"
190 PRINT @296,"E=EDITAR LINHA"
R$R$TAB(8)"D=APAGAR LINHA"R$R$T
AB(8)"L=LISTAR NA TELA"
200 A$=INKEY$:IF A$="" THEN 200
210 JJ=INSTR("GSEDLA",A$)
220 IF JJ=0 THEN PRINT "<"A$">I
NVALIDO":FOR J=1 TO 1000:NEXT:G
OTO 180
230 CLS:ON JJ GOSUB 1420,1450,1
490,1710,1760,280
240 PRINT @0,"PRESSIONE <ENTER>
 PARA CONTINUARQUALQUER OUTRA T
ECLA PARA MENU  PRINCIPAL"
250 A$=INKEY$:IF A$="" THEN 250
260 IF A$<>R$ THEN 180
270 ON JJ GOSUB 1420,1450,1700,
1710,1850,290:GOTO 240
280 K=0:K9=0:P0=0
290 PS=0
300 PS=PS+1:IF PS<4 THEN K=K0:P
=P0:PRINT @1,"INICIO DA ETAPA"P
S:GOTO 330
310 P0=P:RETURN
320 IF PS=3 THEN PRINT " ERRO D
E POSBYTE"
330 GOSUB 1320
340 GOSUB 1260:OP$=C$:IF LEFT$(
OP$,1)="*" AND PS=3 THEN PRINT
OP$
350 IF LEFT$(OP$,1)="*" THEN 33
0
360 IF OP$="END" AND PS=3 THEN
PRINT:PRINT"    FIM.ULTIMO ENDE
RECO";P-1
370 IF OP$="END" THEN 300
380 IF OP$<>"ORG" THEN 420
390 GOSUB1260:S=0:IF LEFT$(C$,1
)="*" THEN S=P:C$=MID$(C$,2)
400 P=VAL(C$)+S:IF PS=3 THEN PR
INT:PRINT"      ORG";P
410 GOTO 330
420 IF P=0 AND PS=3 THEN PRINT"
 FALTA ORG":P=35000
430 CC=0:DF=0
440 C=INSTR(SK$(CC),OP$):IF C<>
0 THEN GOSUB 530:GOTO 570
450 C=INSTR(SK$(CC),LEFT$(OP$,3
)):IF C<>0 AND RIGHT$(OP$,1)<"C
" AND MID$(SK$(CC),C+3,1)<"A" T
HEN GOSUB 530:GOTO 540
460 IF C<>0 AND RIGHT$(OP$,1)="
B" GOSUB 530:GOTO 560
470 C=INSTR(SK$(CC),RIGHT$(OP$,
3)):IF C<>0 AND LEFT$(OP$,1)="L
" GOSUB 530:GOTO 550
480 CC=CC+1:IF CC<2 THEN 440
490 IF PS=3 THEN PRINT OP$
500 GOSUB 1350:Q3=Q2:RR(Q2)=P:I
F C$="" THEN 340
510 IF PS=3 THEN PRINT " LINHA
NAO RECONHECIDA"
520 GOTO 330
530 C=VAL(MID$(SK$(CC),C-2,2)):
RETURN
540 OP=K1(C)-32+16*(RIGHT$(OP$,
1)="A"):K2=0:GOTO 580
550 OP=K1(C)+4096:K2=5:GOTO 580
560 OP=K1(C)+64:K2=1:GOTO 580
570 OP=K1(C):K2=K2(C)
580 IF PS=3 THEN BY=P/256:GOSUB
 1240:BY=P-32768:GOSUB 1240:PRI
NT" "OP$;
590 IF OP>-1 THEN 740
600 GOSUB 1260:AD$=C$:IF PS=3 T
HEN PRINT " " LEFT$(AD$,10;
610 ON -OP GOTO 620,630,660,730
620 GOSUB 1860:IF NU=0 THEN RR(
Q3)=R:GOTO 330 ELSE 1050
630 IF PS=3 THEN PRINT TAB(20);
640 GOSUB 690:BY=255ANDR:GOSUB
1230
650 IF B$=AD$ THEN 330 ELSE 640
660 IF PS=3 THEN PRINT TAB(20);
670 GOSUB 690:BY=INT(R/256):GOS
UB 1230:BY=R-256*BY:GOSUB 1230
680 IF B$=AD$ THEN 330 ELSE 670
690 NN=INSTR(AD$,","):IF NN=0 T
```

```
HEN B$=AD$:GOTO 710
700 B$=LEFT$(AD$,NN-1):AD$=MID$
(AD$,NN+1)
710 IF LEFT$(B$,1)="$" THEN R=V
AL("&H"+MID$(B$,2))ELSE R=VAL(B
$)
720 RETURN
730 GOSUB 1860:IF NU=0 THEN P=P
+R:GOTO 330 ELSE 1050
740 B=239:IF K2=0 THEN 1140
750 GOSUB 1260:AD$=C$:IF PS=3 T
HEN PRINT " " LEFT$(AD$,9);
760 IF K2>3 THEN 1040
770 IF(K2<>3 OR (LEFT$(OP$,2)="
LD")) AND LEFT$(AD$,1)="#" THEN
 AD$=MID$(AD$,2):DF=1:OP=OP-48:
GOTO 1040
780 IF RIGHT$(AD$,1)="]" THEN A
D$=MID$(AD$,2,LEN(AD$)-2):B=B+1
6
790 IF OP<52 OR OP>55 THEN 870
800 K2=1:R=0:AA$=AD$
810 NN=INSTR(AA$,","):IF NN=0 T
HEN U$=AA$:GOTO 830
820 U$=LEFT$(AA$,NN-1):AA$=MID$
(AA$,NN+1)
830 IF U$=RIGHT$(OP$,1)THEN 320
840 NN=-1:FOR J=0 TO 9:IF U$=PU
$(J) THEN NN=J
850 NEXT:IF NN<0 THEN 320
860 R=R ORPU(NN):IF U$=AA$ THEN
 1140 ELSE 810
870 K2=3:C=INSTR(AD$,","):IF C=
0 THEN 1040
880 IF OP<>30 AND OP<>31 THEN 9
50
890 NN=INSTR(AD$,","):U$=LEFT$(
AD$,NN-1):GOSUB 930:N1=H-1:IF H
=0 THEN 320
900 U$=MID$(AD$,NN+1):GOSUB 930
:N2=H-1:IF H=0 THEN 320
910 IF(8ANDN1)<>(8ANDN2) THEN 3
20
920 R=16*N1+N2:K2=1:GOTO 1140
930 FOR J=1 TO 12:IF Y$(J)=U$ T
HEN H=J
940 NEXT:RETURN
950 C2=C+1:FOR J=0 TO 12:L=LEN(
X$(J)):IF MID$(AD$,C2,L)<>X$(J)
 THEN 980
960 IF (B ORV(J)) AND 239<239 T
HEN 320
970 C2=C2+L:B=B AND V(J):IF KK(
J) THEN K2=KK(J)-1
980 IF J=9 THEN J2=C2:C2=C2+(C2
-1)*(K2<6)
990 NEXT
1000 IF(15 AND B)=15 THEN B=B-6
1010 IF PS=3 AND J2<=LEN(AD$)AN
D K2<>7 THEN PRINT "ERRO DE IND
EXACAO":GOTO 330
1020 IF(K2=0 AND C>2) OR (MID$(
AD$+",",J2,1)<>",") AND PS=3 TH
EN PRINT "ERRO DE ENDERECAMENTO
":GOTO 330
1030 AD$=LEFT$(AD$,C-1):IF J2=C
 THEN R=0:GOTO 1060
1040 GOSUB 1860:IF NU=0 THEN 10
60
1050 IF PS=3 THEN PRINT" ENDERE
CAMENTO NAO ENTENDIDO"
1060 IF K2=7 THEN K2=3:GOTO 108
0
1070 IF K2>3 THEN R=R-P-2+(OP>2
55)+(B<>239)+(K2>4):R=R-((K2=6)
AND(R>-129)AND(R<128)):K2=K2-3
1080 IF B=239 AND K2=3 THEN K2=
2:IF R<256 AND DF=0 THEN K2=1:O
P=OP-32:IF(240 AND OP)=80 THEN
OP=OP-80
1090 IF PS=3 AND((OP>31 AND OP<
48)OR OP=141)AND(R<-128 OR R>12
7)THEN PRINT " DESVIO FORA DE F
AIXA";:GOTO 330
1100 IF B=255 THEN B=159:K2=2
1110 IF B<>239 THEN OP=OP-16:IF
 K2=3 THEN K2=2:IF ABS(R+.5)<12
```

```
8 THEN K2=1:B=B-1:R=255 AND R
1120 IF(15 AND B)=8 AND R=0 THE
N B=B-4:K2=0
1130 IF(31 AND B)=8 AND (R<16 O
R R>239) THEN B=(B-136)OR(31 AN
D R):K2=0
1140 IF PS=3 THEN PRINT TAB(20)
;
```

```
1150 IF OP=>0 THEN BY=OP/256:GO
SUB 1220:BY=OP:GOSUB 1230
1160 IF B<>239 THEN BY=B:GOSUB
1230
1170 IF K2=0 THEN 330
1180 GOSUB 1200:IF K2=2 THEN BY
=R/256:GOSUB 1230
1190 BY=R-256*INT(R/256):GOSUB
1230:GOTO 330
1200 IF PS=3 THEN PRINT " ";
1210 RETURN
1220 IF INT(BY)=0 THEN RETURN
1230 P=P+1:IF PS=3 THEN POKE P-
1,255 AND BY
1240 BY=255 AND BY:IF PS=3 THEN
 PRINT RIGHT$("0"+HEX$(BY),2);
1250 RETURN
1260 IF K>N THEN C$="END":RETUR
N
1270 K1=K9+1:IF K9>=LEN(T$(K))
THEN C$=" FALTA MNEMONICO":RETU
RN
1280 K9=K1:IF MID$(T$(K),K1,1)=
" " THEN 1270
1290 IF K9>LEN(T$(K))THEN C$=MI
D$(T$(K),K1,K9-K1):RETURN
1300 IF MID$(T$(K),K9,1)<>" " T
HEN K9=K9+1:GOTO 1290
1310 C$=MID$(T$(K),K1,K9-K1):RE
TURN
1320 IF K9<=LEN(T$(K))AND PS=3
THEN PRINT RIGHT$(T$(K),LEN(T$(
K))-K9+1);
1330 K=K+1:K9=0:IF PS=3 THEN PR
INT
1340 RETURN
1350 X$=""
1360 IF C$<"A" OR C$>="[" THEN
1380
1370 X$=X$+LEFT$(C$,1):C$=MID$(
C$,2):GOTO 1360
1380 IF C$<>"" THEN RETURN
1390 FOR Q2=1 TO VV:IF X$=Z$(Q2
) THEN 1410
1400 NEXT:VV=VV+1:Z$(VV)=X$:Q2=
VV:RR(VV)=23000
1410 RETURN
1420 CLS:MOTORON:PRINT @161,"PO
SICIONE O GRAVADOR, PRESSIONE Q
UALQUER TECLA E APERTE <PLAY>."
1430 U$=INKEY$:IF U$="" THEN 14
30
1440 OPEN"I",#-1,"ASM":PRINT" C
ARREGANDO PROGRAMA":INPUT #-1,N.
:FOR J=1 TO N:INPUT #-1,T$(J):N
EXT:CLOSE #-1:RETURN
1450 CLS:MOTORON:PRINT @161,"PO
SICIONE O GRAVADOR , APERTE    <
RECORD> E PRESSIONE QUALQUER
TECLA."
1460 U$=INKEY$:IF U$="" THEN 14
60
1470 OPEN"O",#-1,"ASM":PRINT" G
RAVANDO PROGRAMA":PRINT #-1,N:F
OR J=1 TO N:PRINT #-1,T$(J):NEX
T:CLOSE #-1:RETURN
1480 PRINT #-1,N:FOR J=1 TO N:P
RINT#-1,T$(J):NEXT:CLOSE#-1:RET
URN
1490 PRINT " INTRODUZA O NUMERO
 DA LINHA (LI NHAS NUMERADAS DE
 DEZ EM DEZ)"
1500 INPUT K:CLS
1510 K2=K/10:IF K2>N THEN K2=N+
1:N=N+1:T$(K2)="":PRINT @480,""
1520 IF K2<.1 THEN K2=.1
1530 IF K2=INT(K2) THEN 1550
1540 K2=INT(K2)+1:FOR K3=N TO K
2-1 STEP -1:T$(K3+1)=T$(K3):NEX
T:N=N+1:T$(K2)=""
1550 P1=1478:P0=P1
1560 PRINT @448,K;TAB(6)T$(K2):
P9=P0+LEN(T$(K2))
1570 IF P1<P0 THEN P1=P0
1580 IF P1>P9 THEN P1=P1-1
1590 P8=PEEK(P1):POKE P1,63 AND
 P8
1600 P7=0:A$=INKEY$:IF A$="" TH
EN 1600
1610 IF A$=R$ THEN POKE P1,P8:R
ETURN
1620 IF A$=CHR$(9) THEN POKE P1
,P8:P1=P1+1:GOTO 1580
1630 IF A$=CHR$(8) THEN POKE P1
,P8:P1=P1-1:GOTO 1570
1640 IF A$=CHR$(10) THEN A$="":
GOTO 1670
1650 IF A$=CHR$(94) THEN A$=" "
+MID$(T$(K2),P1-P0+1,1):P7=-1:G
OTO 1670
1660 IF A$<" " THEN 1600
1670 IF P1-P0+1>LEN(T$(K2)) THE
N T$(K2)=LEFT$(T$(K2),P1-P0)+A$
:GOTO 1690
1680 T$(K2)=LEFT$(T$(K2),P1-P0)
+A$+RIGHT$(T$(K2),LEN(T$(K2))-P
1+P0-1)
1690 P1=P1-(LEN(A$)>0)+P7:GOTO
1560
1700 PRINT @32,"":K=K+10:GOTO 1
510
1710 IF N=0 THEN CLS:PRINT" NAD
A A APAGAR":FOR C=1 TO 1000:NEX
T:RETURN
1720 CLS:PRINT"INTRODUZA NO DA
LINHA            (LINHAS NUMERAD
AS DE DEZ EM DEZ)"
1730 INPUT K:K2=K/10
1740 IF K2>N OR K2<1 OR K2<>INT
(K2) THEN PRINT" ESTA LINHA NAO
 EXISTE":RETURN
1750 K=K2:FOR K3=K2 TO N:T$(K3)
=T$(K3+1):NEXT:N=N-1:PRINT @95,
K*10;"  ";T$(K):RETURN
1760 IF N=0 THEN PRINT " NADA A
 LISTAR":FOR C=1 TO 1000:NEXT:R
ETURN
1770 PRINT" INTRODUZA  O NO DA
PRIMEIRA E  DA ULTIMA LINHA  (L
INHAS NUMERA-DAS EM   MULTIPLOS
 DE 10)"
1780 INPUT K,K2:K=INT(K):K2=INT
(K2):K1=K/10:K2=K2/10
1790 IF K2>N THEN K2=N
1800 IF K1<1 THEN K1=1
1810 IF K2<K1 AND K2=N THEN RET
URN
1820 IF K2<K1 THEN CLS:PRINT "
CONJUNTO DE LINHAS INVALIDO":GO
TO 1770
1830 CLS:PRINT @96,;:FOR K3=K1
TO K2:PRINT K3*10" "T$(K3):NEXT
1840 RETURN
1850 K=K2-K1:K1=K2+1:K2=K1+K:IF
 N=0 THEN 1760 ELSE 1790
```

```
1860 NU=0:R=0
1870 S=1
1880 IF AD$="" THEN RETURN
1890 X$=LEFT$(AD$,1):BD$=MID$(A
D$,2):IF X$="*" THEN R=R+P*S:AD
$=BD$:GOTO 1870
1900 IF X$="+" THEN AD$=BD$:GOT
O 1880
1910 IF X$="-" THEN AD$=BD$:S=-
S:GOTO 1880
1920 Q=0:IF X$<>"%" THEN 1950
1930 IF BD$>="0" AND BD$<"2" TH
EN Q=Q*2+ASC(BD$)-48:BD$=MID$(B
D$,2):GOTO 1930
1940 R=R+Q*S:AD$=BD$:GOTO 1870
1950 IF X$<>"$" OR BD$<"0" OR B
D$>"F" THEN 1980
1960 Q2=VAL("&H"+LEFT$(BD$,1)):
BD$=MID$(BD$,2):Q=Q*16+Q2:X$=LE
FT$(BD$,1)
1970 IF(X$>"/" AND X$<":") OR (
X$>"@" AND X$<"G") THEN 1960 EL
SE R=R+Q*S:AD$=BD$:GOTO 1870
1980 IF X$<"A" OR X$>"Z" THEN 2
010
1990 C$=AD$:GOSUB 1350:IF C$<>"
" GOSUB 1390
2000 R=R+RR(Q2)*S:AD$=C$:GOTO 1
870
2010 IF X$<"0" OR X$>"9" THEN R
=0:NU=1:RETURN
2020 IF AD$>"/" AND AD$<":" THE
N Q=Q*10+ASC(AD$)-48:AD$=MID$(A
D$,2):GOTO 2020
2030 R=R+S*Q:GOTO 1870
```

COMO FUNCIONA O PROGRAMA

Não se esqueça de reservar um espaço suficiente de memória para seu programa, usando **CLEAR** antes de rodá-lo.

Para introduzir o seu programa pressione a tecla E. Ele pedirá então um número de linha. Neste programa cada instrução em mnemônico deve ser introduzida em uma linha de BASIC. É preciso também que os números de linha sejam múltiplos de 10 e cada linha contenha apenas um comando Assembly com seus respectivos operandos.

A primeira linha deve abrigar o endereço inicial da porção da memória onde será colocada a rotina em código. É necessário que esse endereço esteja na área reservada por **CLEAR**. Para especificar sua origem, usamos o comando **ORG** seguido do endereço inicial.

Se um endereço inicial não for especificado, o programa tentará posicionar o Assembler a partir do endereço 35000, que pertence à memória ROM. O resultado não será satisfatório, uma vez que não se pode colocar códigos na ROM. O programa procede assim para evitar que o Assembler seja colocado em uma área prejudicial ao funcionamento da máquina. Quando o endereço acima é usado, o programa comunica: "origem não encontrada".

Normalmente, são utilizados os mnemônicos padrão do 6809. Números hexadecimais devem ser precedidos de **$** (cifrão), números binários, de **%** (porcento). Algarismos sem prefixo são considerados decimais.

Alguns Assembler para o TRS-Color reconhecem códigos ASCII, se estes forem precedidos de ' (apóstrofo) ou **!** (ponto de exclamação). Não é o caso de nosso programa. Assim, o comando **FCC**, que manipula caracteres ASCII, não pode ser usado.

Para editar linhas, empregue as teclas do cursor. As setas "direita" e "esquerda" movimentam o cursor ao longo da linha; a seta "para cima" insere e a seta "para baixo" apaga.

A última linha do programa deve ser **END**; o Assembler criará uma linha desse tipo, caso esqueçamos de fazê-lo.

Se, depois de digitarmos uma linha, pressionarmos outra tecla que não **ENTER**, o programa voltará ao menu. Se então pressionarmos L, os mnemônicos serão listados para uma conferência.

Caso haja algum erro, retorne ao menu e pressione E. Especifique então o número da linha a ser corrigida.

Se quisermos inserir uma nova linha entre duas já existentes, devemos entrar no modo de edição e dar a ela um número intermediário. Uma nova listagem mostrará a nova linha no lugar correto, sendo que todas as linhas terão números múltiplos de 10. Apenas uma linha de cada vez pode ser inserida dessa maneira.

Para apagar uma linha, retorne ao menu e pressione D. Especifique então o número da linha a ser apagada. Quando não houver mais modificações a fazer, retorne ao menu e pressione A. O programa em Assembly será então "montado" na memória. Quando o processo terminar, o endereço final do programa em código será mostrado na tela.

A opção de gravação — tecla S — grava apenas o programa em Assembly ou *programa fonte*. Para gravar o Assembler, use o caminho normal. Para gravar o programa em código — ou *programa objeto* — é necessário sair do Assembler usando a tela **BREAK**:

```
CSAVEM "NOME", INÍCIO, FIM,
DEFEXEC
```

NOME é a denominação do programa; **INÍCIO** é o endereço inicial definido por **ORG**; e **FIM**, o endereço final fornecido pelo programa após a montagem.

O último número, **DEFEXEC**, diz ao TRS-Color por onde começar a executar a rotina em código. Geralmente, ele é igual ao endereço inicial. Agora podemos montar qualquer programa a partir de sua listagem em Assembly.

COMO ENCONTRAR ERROS EM PROGRAMAS LONGOS

Mesmo o programador mais experiente terá problemas em digitar programas longos como esse Assembler. Não importa a destreza de seus dedos: fatalmente, em algum lugar, um erro será cometido.

Muitos desses erros são encontrados quando se confere a listagem. As mensagens de erro do micro ajudarão também, se procurarmos saber o que significam.

Tais mensagens, contudo, podem não ser suficientes para detectar o local de um erro em um programa muito longo. Nesses programas, uma linha pode ser executada sem problemas muitas vezes, só vindo a provocar um erro quando uma variável assumir um valor incompatível devido a um erro de digitação cometido em outra parte do programa.

Felizmente, o TRS-Color conta com uma função de rastreamento — *trace* —, que facilita a localização de erros.

Tal função é ativada pelo comando **TRON**, que deve ser usado no modo imediato, sem número de linha. Quando rodamos o programa, ela nos mostra o número da linha que está sendo executada.

Para desativar a função de rastreamento, use o comando **TROFF**.

UM TESTE

Para testar o Assembler, digite o programa de deslocamento da tela para a direita apresentado na página 219. Esse programa resultará nos seguintes códigos:

```
8E 06 00 E6 82 34 04 C6 1F A6
82 A7 01 5A 26 F9 35 04 E7 84
8C 04 00 2E EA 39
```

Em algumas das máquinas compatíveis com o TRS-Color é possível aumentar a velocidade do programa acrescentando **POKE 65495,0** no início da linha 180. Se isto for feito, devemos modificar outras linhas a fim de que a alta velocidade não prejudique a gravação em fita. Assim, adicione **POKE 65494,0** ao início das linhas 1420 e 1450.